MASTER EN SCIENCES INFORMATIQUES

Contribution à la mise en œuvre d'un réseau local

Lemal, Eric

Award date:
1982

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A LA MISE EN OEUVRE D'UN RESEAU LOCAL

Promoteur : Ph. VAN BASTELAER

Mémoire présenté par

Eric LEMAL

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1981-1982

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A LA MISE EN OEUVRE D'UN RESEAU LOCAL

Promoteur : Ph. VAN BASTELAER

Mémoire présenté par

Eric LEMAL

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1981-1982

PLAN GENERAL

PREMIERE PARTIE

RESEAUX ET RESEAUX LOCAUX

DEUXIEME PARTIE

QUELQUES EXEMPLES DE RESEAUX LOCAUX

TROISIEME PARTIE

CONCEPTION DES RESEAUX LOCAUX

QUATRIEME PARTIE

MISE EN OEUVRE D'UN PROTOCOLE DE COMMUNICATION

CINQUIEME PARTIE

MISE EN OEUVRE D'UN RESEAU LOCAL

Je remercie toutes les personnes qui d'une façon quelconque ont participé à la réalisation de ce mémoire

Je tiens en particulier à exprimer ma reconnaissance à Monsieur Ph. Van Bastelaer, promoteur de ce mémoire, pour les conseils apportés lors de la rédaction.

Je tiens à témoigner ma gratitude à Monsieur V. Cordonnier qui m'a acceuilli pour un stage à la faculté d'informatique de l'Université de Lille, ainsi qu'à l'ensemble des membres du laboratoire d'architecture pour leur acceuil au sein de leur département.

Je tiens également à témoigner ma gratitude à Monsieur J.D. Nicoud et R. Sommer ainsi qu'à l'ensemble des membres du laboratoire de micro-informatique de l'Ecole Polytechnique Fédérale de Lausanne, de m'avoir acceuilli lors de ma visite.

Je remercie tout spécialement Monsieur R. Sommer pour l'extrême disponibilité dont il a fait preuve à mon égard.

Je tiens enfin à remercier Monsieur J.P. Adans pour l'aide constante qu'il m'a accordée pendant l'élaboration de ce travail.

INTRODUCTION

INTRODUCTION

La télèinformatique, qui constitue une branche importante de l'informatique, èvolue encore fortement. Dans les premiers systèmes télèinformatiques, les terminaux ont êtê reliês directement à l'ordinateur central. On a ensuite introduit des concentrateurs reliês localement aux terminaux par des lignes lentes, et reliês à distance à un ordinateur central par des lignes rapides. Une troisième êtape a êtê franchie avec l'apparition des frontaux assurant la gestion des transmissions pour le compte de l'ordinateur central. Finalement, nous abordons aujourd'hui une nouvelle êtape dans l'èvolution des architectures des systèmes télèinformatiques avec l'introduction des rèseaux de transmission.

Des besoins de communication, en vue du partage de donnèes, de programmes ou tout simplement de ressources, entre systèmes autonomes se font de plus en plus ressentir.

L'amèlioration des techniques en matière de télècommunication, la tendance marquèe vers une standardisation et la diminution des coûts de la technologie, notamment l'apparition des micro-processeurs, ont provoquê une croissance rapide des rèseaux d'ordinateurs. On assiste à une explosion des publications dans ce domaine et plusieurs constructeurs proposent dèjà leur propre rèseau : DSA d'Honeywell Bull, DNA de Digital, SNA d'IBM, ...

Avec l'amèlioration des techniques de communication à grande vitesse et sur de courtes distances, qui coïncide d'ailleurs avec l'apparition des micro-processeurs, il est devenu possible de rèpartir le travail d'une grosse machine entre plusieurs machines plus petites ; c'est l'informatique distribuèe, qui consiste à donner davantage d'autonomie aux diffèrents èquipements constituant un système.

La tendance actuelle de la bureautique repose sur la possibilitê de communication entre diffèrents èquipements, dans le but de rèaliser des applications informatiques distribuèes (Par exemple le courrier èlectronique). Le rèseau local, lui, permet le partage d'un processeur unique et de ressources chères par plusieurs ordinateurs localisès.

Le but de ce mémoire consiste en l'étude des possibilités de réalisation d'un réseau local. Un réseau d'ordinateurs fait appel à des concepts globaux d'architecture de réseau. La première partie de ce travail s'efforcera de présenter, de manière succincte, les notions essentielles d'architecture des réseaux d'ordinateurs pour aborder par après celles des réseaux locaux.

Ensuite, on analysera de façon plus précise, les différentes techniques utilisées en matière de réseaux locaux. Ce sera le but de la deuxième partie.

La troisième partie, dans un monde où l'on cherche de plus en plus à rationaliser les éléments, se penchera sur les problèmes de standardisation des architectures de réseaux locaux en tentant d'y appliquer le modèle des travaux de l'International Standards Organization (ISO) sur les architectures en couches des réseaux. Elle fera apparaître aussi les problèmes de décisions qui existent au niveau du choix d'une architecture d'implémentation entre composants logiques (software) ou physiques (hardware).

Enfin, le but de ce mémoire étant l'étude des possibilités de réalisation d'un réseau local, les deux dernières parties du travail concernent directement les réalisations d'un logiciel de communication (Quatrième partie) et d'un prototype de réseau local au sein de l'Institut d'Informatique des FNDP ; ce réseau local permettra d'interconnecter à coût réduit, les équipements dont celui-ci dispose déjà (Cinquième partie).

PREMIERE PARTIE

RESEAUX ET RESEAUX LOCAUX

INTRODUCTION

On rencontre des problèmes de communication à plusieurs niveaux dans les ordinateurs:

a. On en rencontre au sein même de la machine dans les moyens de communication qui relient les différentes unités qui la composent (Unité arithmétique, Unité de contrôle, mémoires, processeur, unité d'échange, etc.)

b. On en rencontre à un autre niveau entre les différents éléments qui composent un système informatique : machine centrale, terminaux, périphériques, etc.

c. On en rencontre aussi à un niveau supérieur lorsqu'il s'agit de relier plusieurs systèmes informatiques entre eux pour créer un réseau d'ordinateurs.

L'évolution rapide de la technologie d'aujourd'hui amène sans cesse de nouvelles méthodes de résolution des problèmes de communication. On voit, par exemple, apparaître des mémoires circulantes [CORDONNIER, PETITPREZ] qui concurrencent désormais les "bus" classiques internes aux machines. On voit apparaître les fibres optiques qui remplacent les lignes traditionnelles qui permettent d'interconnecter différents équipements entre eux. On assiste aussi au développement croissant de réseaux internationnaux. Dans ce domaine, l'avènement des microprocesseurs a ouvert une nouvelle voie : les réseaux locaux.

L'objectif de cette première partie sera donc de tracer les grandes lignes de ce qu'on appelle réseaux d'ordinateurs et réseaux locaux d'ordinateurs; le but n'étant pas d'en faire une description complète mais de rappeler les caractéristiques essentielles de leur architecture.

Chapitre 1

1. NOTION DE RESEAU D'ORDINATEURS

1.1. Définitions

Un réseau d'ordinateurs est constitué d'un ensemble de stations qui communiquent entre elles au travers d'un réseau de transport. Ce réseau de communication est constitué d'un ensemble de noeuds connectés entre eux au moyen de lignes de transmission.

Par station, on désignera tout équipement utilisateur de réseaux et capable de traiter et de stocker de l'information, c'est-à-dire des ordinateurs, des systèmes informatiques, des terminaux intelligents ou non, etc.

Un noeud est une unité tel un ordinateur, un concentrateur, un controlleur ou un multiplexeur ..., dont le but exclusif est d'assurer l'acheminement des informations vers un autre noeud, soit directement, soit par l'intermédiaire d'autres noeuds.

Les lignes de transmission sont des moyens de transmission utilisés pour véhiculer les informations ; ce peut être des lignes téléphoniques, des câbles, les voies hertziennes, etc. qui n'incluent pas de fonction de stockage ou de traitement de l'information.
Une ligne de transmission qui relie deux noeuds et seulement deux, est dite ligne bipoint. Par extension, une ligne de transmission qui relie plus de deux noeuds est dite ligne multipoint ou à accès partagé.

1.2. Topologie des réseaux de communication

La topologie d'un réseau est le modèle d'interconnexion des différents noeuds qui le composent.

La taxonomie de Shoch [SHOCH 2] distingue cinq grandes topologies différentes:

- Rèseau en ètoile,
- Rèseau en anneau ou en boucle,
- Rèseau maillè,
- Rèseau bus,
- Rèseau radio.

Voici de manière succincte une description de chacune de ces topologies, pour lesquelles on donnera d'abord le principe d'architecture, les avantages et les inconvènients que l'architecture apporte, puis quelques exemples :

1.2.1. Rèseau en ètoile

Dans un rèseau en ètoile, chaque noeud est directement connectè à un seul commutateur central.

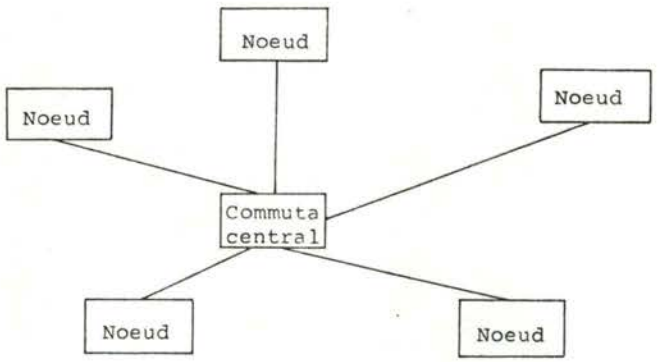


Fig. I.1 Rèseau en ètoile.

Les messages èchangès par deux noeuds transitent toujours par le commutateur central ; celui-ci devient alors source et destination apparente de tous les messages, et peut dès lors être considèrè comme moyen de transport partagè entre les noeuds. C'est lui qui assure le routage des messages entre tous les noeuds du rèseau.

Avantages

Une telle solution est simple à rèaliser.
Elle est aussi efficace pour autant que le nombre de stations ne soit pas trop èlevè, et que les messages èchangès soient courts.

Inconvènients

Tous les messages transitant par le central, un tel système tend rapidement vers une saturation au fur et à mesure que le nombre de stations ou le le nombre de communications

augmente.
Un tel système n'est pas à l'abri d'une défaillance du commutateur central dont il faut d'ailleurs disposer dès le début du système, ce qui nécessite un investissement initial plus élevé.

Exemples

Un exemple typique est la connexion de terminaux à un ordinateur.

1.2.2. Réseau en anneau

Un réseau en anneau est un réseau fermé, constitué d'une seule maille, dans lequel chaque noeud n'est connecté qu'à chacun de ses voisins directs.

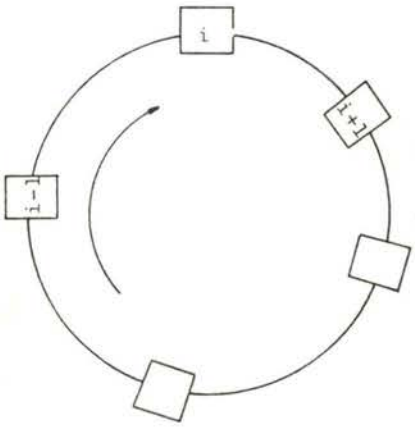


Fig. I.2 Réseau en anneau.

A cause du support physique et de sa configuration, la transmission se fait dans un seul sens. En effet, une transmission bidirectionnelle provoquerait l'arrivée à destination de deux copies du même message, mais à des moments très probablement différents à cause des délais de transmission sur les deux chemins parcourus.

Tout noeud recevant un message qui n'est pas destiné à la station qui lui est connectée, transmet ce message au noeud suivant. Par contre, un noeud qui reçoit un message destiné à la station qui lui est connectée, extrait ce message du réseau.
En d'autres termes, un message passe de noeud en noeud au travers de moyens unidirectionnels.

Ce type de topologie n'implique aucune décision de routage, le noeud expéditeur (i) transmettant simplement son message au noeud voisin (i+1); ce message transite sur l'anneau de noeud en noeud, appelés noeuds répétiteurs, jusqu'à ce qu'il atteigne le noeud destinataire (j).

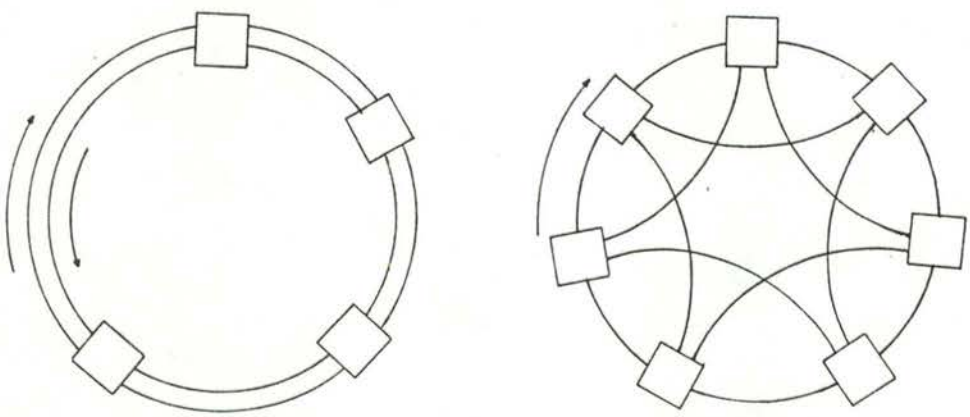
Ce principe requiert de chaque noeud qu'il soit capable de reconnaître un message qui lui est destiné, et qu'il fonctionne en permanence.

Avantages

Dans un anneau, les coûts sont proportionnels au nombre de noeuds à relier.
C'est une solution facile à réaliser, chaque noeud ne connaissant que ces deux voisins adjacents.

Inconvénients

Un des inconvénients majeurs d'une pareille solution provient du fait que chaque noeud joue un rôle de répétiteur et que si l'un d'entre eux ne peut plus assurer son rôle, par suite d'une défaillance, tout le réseau est paralysé. Ceci a conduit à la constitution d'anneaux doubles opérant soit tous les deux dans le même sens, soit en sens opposé, ou à des anneaux "tressés" (braided) dont le second ne relie qu'un noeud sur deux ou plus.



(Anneau double) (Anneau tressé)

Fig. I.3 Réseau en anneau : anneau double et anneau tressé.

Ces systèmes permettent alors de court-circuiter un noeud en panne ou une ligne cassée entre deux noeuds.

Souvent, les réseaux en anneau exigent la présence d'un noeud maître qui contrôle à lui seul tout le réseau : certains auteurs [VAN REMORTEL] préfèrent alors appeler les réseaux où le contrôle est ainsi centralisé, "réseaux en boucle".

Exemples

Anneau de LIU, boucle de PIERCE [LIU 2, PENNEY].

1.2.3. Réseau maillé

Un réseau maillé est un réseau dont chacun des noeuds peut être connecté, à la fois, à plusieurs autres noeuds du réseau ; il se caractérise donc par l'existence possible de plusieurs chemins de communication entre deux noeuds.

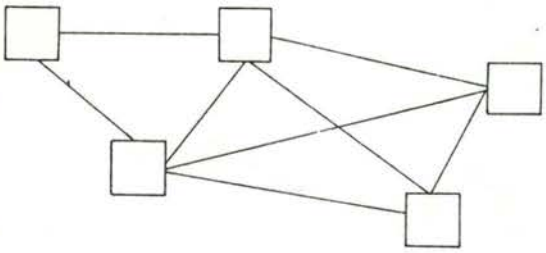


Fig. I.4 Réseau maillé.

Chacun des noeuds doit être capable d'assurer, dans les meilleures conditions, le routage d'un message qui ne lui est pas destiné. Ceci nécessite des algorithmes de routage et de contrôle de flux assez complexes.

Avantages

Dans un réseau maillé, la défaillance d'un noeud n'a pas d'influence dramatique sur le comportement global du réseau : les algorithmes de routage doivent simplement tenir compte de ce qu'un noeud est momentanément hors d'usage.

Inconvénients

Un des principaux inconvénients des réseaux maillés est la complexité des algorithmes de routage.
De plus, l'adjonction d'un n ième noeud est assez coûteuse puisqu'elle peut nécessiter jusqu'à n-1 nouvelles connexions avec les autres noeuds, ainsi que des modifications dans les logiciels de communication de tous les autres noeuds.

Exemples

ARPANET : les noeuds sont constitués d'interfaces processeurs de messages (IMP : Interface Messages Processor) auxquels sont connectés plusieurs stations ou hôtes (Host), ce qui leur donne en fait l'aspect d'une configuration étoilée. Le traffic qui circule entre deux hôtes connectés au même noeud, ne circule pas sur le réseau et est appelé

traffic incestueux.

DECNET : ici, le logiciel des opérations de contrôle et de routage propre au réseau "tourne" sur la même machine que le logiciel d'application de la station. On peut dire que le noeud et la station ont été intégrés dans la même machine.

1.2.4. Réseau bus

A la différence de toutes les topologies précédentes, le réseau bus a une structure linéaire (ouverte).

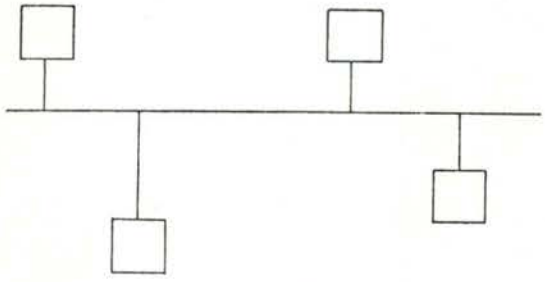


Fig. I.5 Réseau bus.

La transmission y est bidirectionnelle et du type de la diffusion : tous les noeuds "écoutent" le bus et seul répond celui à qui le message est destiné.
Il n'y a donc aucun problème de routage dans ces architectures. Par contre, on y trouvera des conflits d'accès au bus, ce qui oblige soit à définir une politique d'accès par priorité, soit à admettre que des messages puissent rentrer en collision : on parlera alors de gestion de contention.

Avantages

Les coûts sont proportionnels au nombre de stations à connecter.
La défaillance d'une station n'a aucune incidence sur le comportement du réseau.

Inconvénients

Il n'y a pas, à priori, d'inconvénients majeurs propre à ce genre d'architecture, si ce n'est qu'une défaillance du bus même paralysera tout ou partie du réseau.

Exemples

Ethernet, Cobus, Net/one [DIGITAL, SOMMER 1, UNGERMANN]

1.2.5. Réseau radio

Shoch inclut dans sa taxonomie une cinquième famille basée sur les systèmes radio, où les moyens de transmissions sont les voies hertziennes, bien que cette famille ne constitue pas une architecture en elle-même, mais ressemble plus à un réseau bus par son principe d'accès aux voies de communication.

Exemple

L'exemple le plus connu est le réseau ALOHA de l'Université d'Hawai.

1.2.6. Topologies hiérarchiques

Il n'y a aucunes restrictions à construire, sur base des architectures qui viennent d'être définies, des réseaux à structures hiérarchiques.

Pour des réseaux en anneau, on peut interconnecter un anneau à un autre de manière à constituer des niveaux. De cette manière, le trafic inhérent à un seul niveau ne circule pas sur les autres réseaux, ce qui permet de tirer des avantages du point de vue performances et fiabilités.

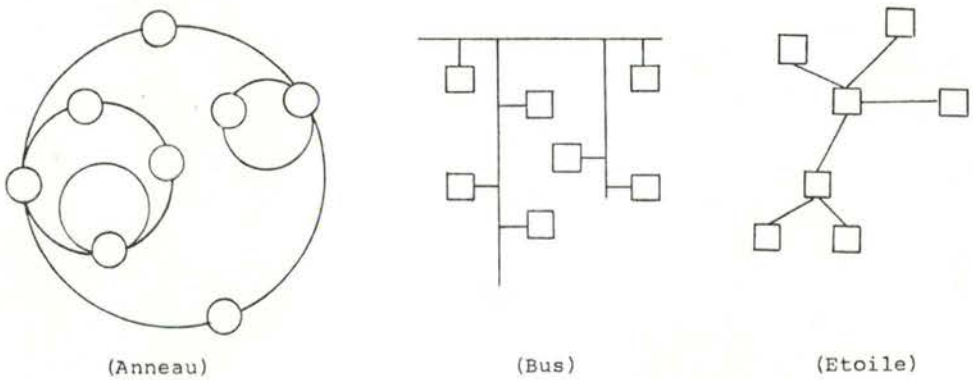


Fig. I.6 Topologies hiérarchisées de réseaux.

De la même manière, on pourra interconnecter entre eux et à différents niveaux des réseaux bus ou en étoile.

1.3. Autres classifications

Il existe des méthodes de classification des réseaux qui sont plus complètes. On citera comme exemple celle d'Anderson [ANDERSON] qui englobe entièrement celle de Shoch.

Le modèle de typologie d'Anderson

Le modèle de typologie d'Anderson se base sur les éléments de quatre niveaux de choix qui permettent d'aboutir à une architecture particulière d'un réseau.

Ces éléments sont par niveau :

1. Stratégie de transfert : le transfert entre deux noeuds peut se faire directement sur un chemin, ou indirectement en transitant par des commutateurs.

2. Méthode de contrôle des transferts : le contrôle des transferts peut être centralisé (un seul noeud contrôle les transferts de tous les messages envoyés) ou distribué (plusieurs noeuds effectuent ce contrôle).

3. Type de chemin de transfert : les chemins de transfert peuvent être de trois types :

   - dédicacés et unidirectionnels de noeud à noeud,
   - dédicacés et bidirectionnels de noeud à noeud,
   - partagés et bidirectionnels entre plusieurs noeuds.

4. Topologie d'interconnexion des noeuds : on y retrouve les topologies classiques, notamment celles définies par Shoch, à savoir bus, anneau, étoile, maille, etc.

Une combinaison des éléments de ces quatre niveaux fournit alors neuf familles différentes de réseaux :

- Anneau à contrôle distribué,
- Anneau à contrôle centralisé,
- Réseau maillé,
- Réseau en étoile,
- Mémoire commune,
- Bus à contrôle distribué,
- Bus à contrôle centralisé,
- Réseau régulier,
- Réseau irrégulier.

Chapitre 2

2. NOTION DE RESEAU LOCAL

2.1. Définitions

On retrouve dans la littérature beaucoup de définitions variées se rapportant aux réseaux locaux. En principe, elles s'accordent toutes pour dire que :

1. Les réseaux locaux forment un sous-ensemble des réseaux d'ordinateurs,

2. On distingue les réseaux locaux par le fait qu'il doivent servir des utilisateurs dans un environnement géographiquement limité [COTTON],

3. Du fait de ces distances limitées, un réseau local doit pouvoir assurer un transfert de données à grande vitesse [COTTON, VAN REMORTEL],

4. Enfin, un réseau local doit être fiable.

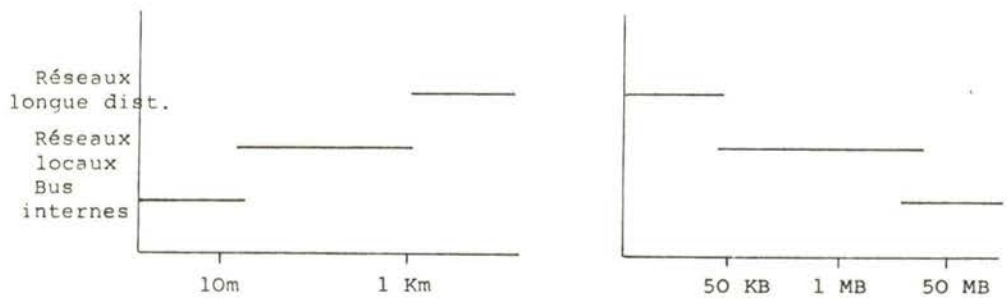


Fig. I.7 Situation des réseaux locaux.

On peut aussi considérer les réseaux locaux comme une classe intermédiaire entre les réseaux longue-distance et les réseaux multiprocesseurs (ou systèmes d'interconnexion entre les processeurs et les unités mémoires, de contrôle et d'entrée/sortie). Les premiers couvrent des distances de l'ordre des cent ou mille kilomètres et ont des vitesses dee transmission de l'ordre des 50 Kb/s. Les derniers couvrent des distances de l'ordre de quelques mètres et atteignent des vitesses de 50 Mb/s.

Le réseau local se trouve justement au milieu et, typiquement, il couvre une distance de l'ordre du kilomètre et atteint des vitesses de transfert de l'ordre d' 1 Mb/s.

2.2. Topologie des réseaux locaux

La taxonomie de Shoch s'adapte très bien aux réseaux locaux. Elle est cependant trop complète.

C'est pourquoi, on en retiendra principalement les topologies suivantes :

- Réseau en étoile,
- Réseau en anneau,
- Réseau bus.

Des exemples concrets de réseaux locaux relatifs à ces architectures sont donnés dans la deuxième partie de ce mémoire.

2.3. Avantages des réseaux locaux

On retrouve dans la littérature pas mal de discussions quant aux avantages qu'apportent les réseaux locaux [LOVELAND, MONNIER, NICOUD, VAN REMORTEL].
De façon générale, on cite le plus souvent :

- Le caractère modique des coûts d'implantation d'un réseau local,
- L'amélioration dans l'utilisation des ressources de ces réseaux locaux,
- La possibilité de spécialisation des processeurs,
- La plus grande disponibilité du système,
- La sécurité des réseaux locaux et ses incidences au niveau des utilisateurs.

1. Faible coût

Le coût raisonnable est, de loin, la caractéristique la plus souvent citée quand on parle de réseaux locaux. Ceci est peut-être dû au fait qu'elle est elle-même à la base de l'évolution actuelle des systèmes décentralisés, ce faible coût étant principalement dû à l'essor de la technologie actuelle dans le domaine de l'électronique. De manière générale, il semble que le coût total d'un système formé de plusieurs équipements de faible complexité est moins élevé que le coût d'un seul équipement complexe réalisant les mêmes fonctions que le premier système [BOUHOT].

2. Utilisation optimale des ressources

Les systèmes centralisés se distinguent des systèmes répartis par le travail considérable que doit faire leur système d'exploitation pour en assurer la cohérence.

Puisque les systèmes centralisés font l'objet de requêtes nombreuses et diversifiées, leurs systèmes d'exploitation sont obligés d'effectuer beaucoup de contrôles et de garantir des protections d'accès nécessaires : ce travail peut représenter une charge importante du système, avec pour conséquence la dégradation des performances.

Les systèmes répartis permettent, au contraire, d'attribuer à chacun des processeurs qui le constituent, une fonction bien précise (Gestion d'une banque de données, gestion des impressions, etc.) .

3. Spécialisation des processeurs

Cette division du travail apportée par la répartition, permet à chaque processeur d'être spécialisé et d'être adapté au travail qu'on exige de lui.

Il est certain aussi que cette spécialisation facilite les traitements effectués au niveau local de chaque processeur, ce qui n'aurait pas été le cas dans un système centralisé. On obtient ainsi une structure modulaire qui facilite aussi la mise au point, l'intégration, les modifications ou les extensions de tels systèmes.

4. Qualités des services

La conséquence d'une meilleure adaptation des processeurs aux travaux qui leur sont demandés est l'amélioration des services rendus aux utilisateurs grâce à un temps de réponse minimum, surtout lorsque la majeure partie du traitement peut s'exécuter localement. Chaque élément d'un système réparti ayant une certaine autonomie (Processeur, mémoire, ...) il sera possible d'exécuter différents traitements simultanément.

5. Fiabilité du réseau

La tendance à la répartition contribue aussi à accroître la fiabilité d'un système informatique où on accepte de moins en moins qu'un seul de ses composants n'affecte son comportement global. Une défaillance locale d'un des éléments ne doit pas paralyser l'ensemble d'un réseau : ceci est possible grâce à l'autonomie de ces composants.

6. Sécurité des systèmes répartis et de leurs utilisateurs

A l'heure où l'informatique devient l'élément clé d'une organisation, où on lui confie toutes les données vitales de cette organisation (Gestion financière, brevets, fichiers importants,...) la sécurité d'un système informatique est primordiale, tant du point de vue vulnérabilité du système (Destruction par l'eau, le feu ; incidents techniques, électriques,...) que du point de vue confidentialité des informations que le système traite (Vols, fraudes,...).

La sécurité semble être résolue par le fait de la répartition ; elle débouche directement sur la disponibilité du système.
La confidentialité d'un système réparti provient du fait que certaines informations ne "quittent" jamais le site où elles doivent être traitées.
A l'heure actuelle, les problèmes de la sécurité et de la confidentialité soulèvent beaucoup de controverses [DATTIN].

DEUXIEME PARTIE

QUELQUES EXEMPLES DE RESEAUX LOCAUX

INTRODUCTION

On vient de définir conjointement les caractéristiques essentielles des réseaux locaux (Espace géographique limité, vitesse de tranferts, simplicité, fiabilité et coût réduit) et les topologies possibles qu'ils peuvent adopter (Bus, anneau, étoile, maille).

Il est étonnant de voir, depuis une quinzaine d'années, le nombre d'architectures différentes qui ont été proposées en matières de réseaux locaux, bien que la majeure partie d'entre elles puisse être répartie dans deux catégories distinctes.

Quelles sont ces catégories ? Quelles sont ces architectures et leurs particularités ? Voilà les questions qui seront abordées dans cette partie.

Chapitre 1

1. LES TOPOLOGIES EN PRESENCE

On a abordé dans la première partie, les différentes topologies des réseaux de communication. On a aussi abordé le modèle de taxonomie de Shoch qui apparaît comme trop complet dans le cadre des réseaux locaux.

1.1. Définitions et rappels

Il est peut-être utile de rappeler qu'elles sont les caractéristiques essentielles des réseaux locaux; on en donnera ainsi des définitions "imagées" ainsi que des trois topologies avec lesquelles on travaillera dans la suite:

1. Un réseau local peut être considéré comme un ensemble de noeuds, situés dans un environnement géographiquement restreint, qui communiquent entre eux via des lignes de transmission. Ces noeuds échangent des messages, relativement courts, à des vitesses relativement élevées.

2. Le modèle d'interconnexion de ces noeuds se divise en trois classes distinctes :

   - Les réseaux locaux du type anneau, où chaque noeud est relié à ses deux voisins immédiats par des lignes de communication bipoint.

   - Les réseaux locaux du type étoile, où chaque noeud est directement relié à un unique noeud central qui gère toutes les opérations avec les noeuds satellites.

   - Les réseaux locaux du type bus, où tous les noeuds sont connectés sur une même ligne de communication qu'ils se partagent.

   Les architectures de type hiérarchique relèvent plus d'une interconnexion de plusieurs réseaux locaux distincts et sont classées suivant le type de ceux-ci.

3. Il reste à définir les éléments qui sont le support du dialogue entre les noeuds d'un réseau :

   Un message est un ensemble de caractères de données encadré par deux ensembles de caractères de contrôle, l'entête, et la queue . L' entête permet aux noeuds du réseau de connaître l'expéditeur et le destinataire du message, en plus d'autres renseignements tels la longueur ou le type du message ; la queue contient, elle, les informations nécessaires au contrôle d'erreur.

   Un acquittement, ou accusé de réception, est un ensemble de caractères utilisé pour avertir l'expéditeur que son message a été correctement ou incorrectement reçu. L'acquittement circule donc dans le sens "inverse" du message auquel il est associé.

1.2. Le modèle de Shoch et les réseaux locaux

Ce sont les exigences même d'un réseau local qui éliminent certaines des topologies rencontrées dans le modèle de Shoch.

La simplicité du logiciel de communication va à l'encontre d'un réseau maillé où le contrôle de flux et du routage représentent souvent des traitements très complexes.

La fiabilité et la disponibilité se portent en faveur d'une décentralisation du contrôle des réseaux locaux ; les topologies où ce contrôle est trop centralisé seront donc délaissées au profit des autres.

1.3. Les deux tendances : l'anneau et le bus

Au terme de cette élimination, il subsiste en fait deux topologies principales, l'anneau et le bus, sur lesquelles vont se développer à partir de 1969 des idées de réseaux locaux.

Nous aborderons d'abord les réseaux locaux basés sur la topologie du type anneau, et par la suite ceux basés sur une topologie de type bus.

Chapitre 2

2. RESEAUX EN ANNEAU

Rappelons brièvement en quoi consiste un réseau en anneau : c'est un réseau dont chaque noeud est relié par des moyens unidirectionnels à ses deux voisins directs. Un noeud recevant un message qui ne lui est pas destiné, doit aussitôt retransmettre ce message vers le noeud suivant, et ainsi de suite pour que ce message soit finalement acheminé jusqu'au noeud destinataire.

2.1. L'anneau de FARMER-NEWHALL (1969)

(Technique du jeton)

La première méthode d'utilisation d'un anneau, fut étudiée dès 1969 par FARMER et NEWHALL ; elle est connue sous le nom de technique du drapeau [NEWHALL] ou du jeton [CORNAFION]

Principe

Un jeton est généré dès que l'anneau est mis en marche ; ce jeton est aussitôt attribué à un des noeuds du réseau.

Le noeud qui possède le jeton est maître de l'anneau et peut alors, s'il le veut, envoyer son (ses) message(s) sur l'anneau. Quand il a fini, il relance son jeton sur l'anneau vers le noeud suivant. Si ce dernier désire aussi envoyer un message, il fait de même, sinon il passe le jeton au noeud suivant et ainsi de suite.

En d'autres termes, un jeton circule sur un anneau. Lorsqu'un noeud veut émettre un message, il doit attendre ce jeton ; lorsque celui-ci se présente, le noeud le retire de l'anneau, envoie son message, puis relance le jeton.

Travail des noeuds

Le noeud destinataire prend copie du message envoyé, mais ne l'enlève pas de l'anneau. Le message revient donc vers son propre expéditeur qui peut alors le retirer de l'anneau et vérifier s'il a, ou non, été altéré durant sa transmission. Si nécessaire, le noeud réémettra son message jusqu'à ce que la transmission se soit réalisée sans accroc.
A un instant donné, seuls circulent sur l'anneau les messages en provenance du même noeud.

Format des messages

Les messages qui circulent sur cet anneau sont de longueur variable, c'est-à-dire que la partie "données" peut varier en longueur.
Seul l'entête a toujours même longueur ; il comprend les renseignements suivants :

- Adresse du noeud destinataire,
- Adresse du noeud expéditeur,
- Type du message (Contrôle ou données).

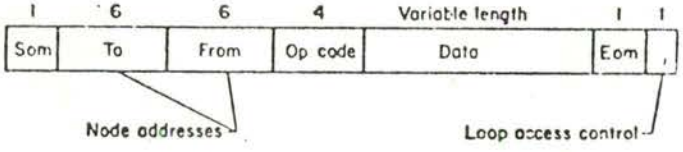


Fig. II.1 Format des messages.

Le jeton n'est, lui, constitué que d'un seul caractère spécial ; ce jeton est accolé au dernier message envoyé par un noeud.

Remarques

1. Un message ne pourra pas "tourner" éternellement sur l'anneau puisque le retrait ne dépend pas de son contenu : l'expéditeur élimine, en effet, tout les messages qu'il a émis.

   Il n'existe cependant pas de mécanisme d'acquittement ; l'expéditeur peut seulement vérifier que le contenu de son message n'a pas été altéré. Ceci ne lui garantit pas que le destinataire ait pris copie du message. Il existe en effet une probabilité non nulle que le destinataire n'ait pas pris copie de ce message.
   De même, il existe une probabilité non nulle que deux corruptions s'annulent : un message peut sembler avoir été correctement reçu par le destinataire, alors qu'il le l'est pas.

   Un message pouvant être corrompu entre le destinataire et l'origine, il faut doter le destinataire d'un système lui permettant de reconnaître un message envoyé pour la première fois

de celui ré-envoyé après détection d'erreurs.

2. En terme de capacité : puisqu'un message n'est retiré que par son expéditeur, 50 % de la capacité de l'anneau sont perdus à cause de la méthode utilisée.

3. Il existe certains risques dans ce genre d'architecture ; notamment qu'un noeud s'accapare le jeton trop longtemps, interdisant ainsi à d'autres noeuds d'envoyer leurs messages, ou que le jeton soit perdu.

   C'est pourquoi l'architecture nécessite un ordinateur superviseur qui se charge des initialisations, du contrôle de flux, de la perte du jeton, etc.

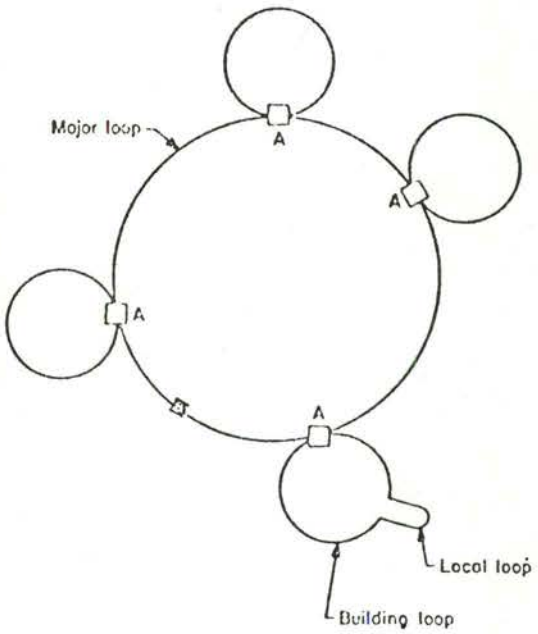


Fig. II.2 L'anneau de FARMER - NEWHALL.

4. L'anneau de Farmer et Newhall a été initialement conçu sous une forme hiérarchique, où une boucle de haut niveau transfère les données entre boucles de bas niveau.

2.2. L'anneau "DCS" de FARBER (1972)

(Technique du jeton)

L'amélioration faite dans l'anneau DCS (Distibuted Computer System) par rapport à la méthode de Farmer et Newhall, vise principalement la fiabilité des transmissions de message : c'est le problème des acquittements qui permettent , à un noeud expéditeur, de se rendre compte dans quelles conditions son message à été reçu, ou

éventuellement n'a pas été reçu, par le destinataire.
Les différences essentielles se situent donc au niveau du format des messages et de leur constitution ; quelques modifications apparaissent aussi au niveau des principes généraux de ce réseau.

Principes

Le principe du DCS (Distributed Computer System) est grosso-modo le même que celui utilisé dans l'anneau de Farmer et Newhall, si ce n'est :

- que les messages ont tous même longueur, fixée et déterminée à l'avance et que les caractères de contrôle, contenus dans l'en-tête et la queue, sont en plus grand nombre pour permettre un meilleur contrôle des transmissions ;

- que les messages sont adressés à des processus et non à une station : chaque noeud contient une table associative qui indique le nom des processus qui s'y déroulent (Jusqu'à 16 processus). Lorsqu'un message passe dans un noeud, la table associative est consultée : si le processus destinataire est présent, le message est copié dans une file d'attente du noeud et des informations d'états sont mises à jour dans le message qui retourne alors vers son expéditeur ;

- qu'un noeud ne peut jamais envoyer qu'un seul message à la fois lorsqu'il a reçu le jeton.

Format des messages

Chaque message contient le nom du noeud d'origine. Ce noeud pourra donc effectuer le retrait du message qu'il a émis après un tour complet dans l'anneau.

Les différences fondamentales entre le format des messages "DCS" et ceux de l'anneau de Farmer et Newhall, consistent dans l'adjonction d'un bit de parité positionné par le destinataire, d'un indicateur de copiage qui indique si le message a été copié ou non, et d'un code de contrôle d'erreur (CRC).

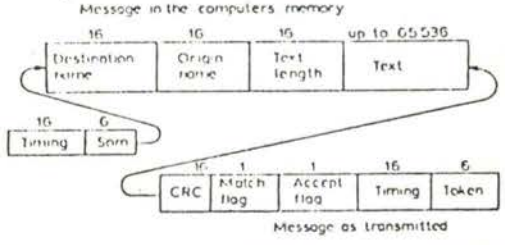


Fig. II.3 Format des messages.

Puisqu'un noeud ne peut jamais émettre qu'un seul message à la fois lorsqu'il a reçu le jeton, il est clair que ce jeton est

toujours accolé au message envoyé.

Remarques

1. Le mécanisme d'acquittement permet à l'expéditeur de savoir non seulement si le message a été copié, mais aussi s'il a été correctement reçu.

2. On n'utilise toujours que 50 % de la capacité de l'anneau puisque les messages ne sont retirés que par leur propre expéditeur, lorsqu'ils y sont de retour après un tour complet.

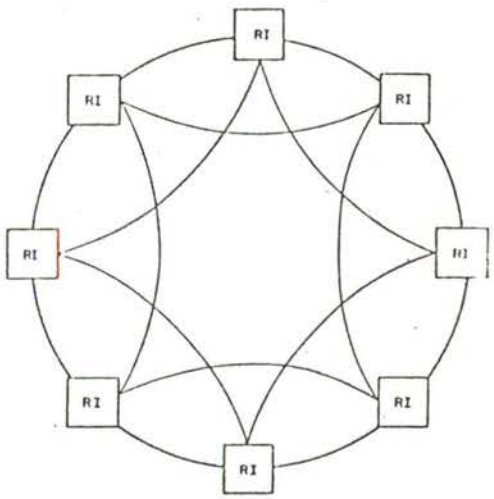


Fig. II.4 L'anneau de FARBER.

3. Il existe toujours un noeud principal qui se charge de la supervision du réseau.

4. Un processus peut "bouger" dans le système sans que d'autres processus n'en soient avertis.

5. La fiabilité du réseau est réalisée grâce à un second anneau (Daisy chain) qui ne relie qu'un noeud sur deux : on obtient ainsi des anneaux tressés (Braided) de sorte que lors d'un mauvais fonctionnement d'un noeud ou d'une ligne, le noeud suivant reçoit ses données depuis la seconde ligne, excluant ainsi de façon logique, le noeud ou la ligne défaillant.

2.3. L'anneau de PIERCE (1972)

(Technique du plateau tournant)

Les études théoriques des réseaux locaux que Pierce a entreprises, ont conduit dès 1972 à la réalisation d'un réseau local basé sur une construction hiérarchique à trois niveaux de plusieurs boucles : on trouve ainsi une boucle nationale unique sur laquelle se greffent des boucles régionales, sur lesquelles se greffent enfin des boucles locales.

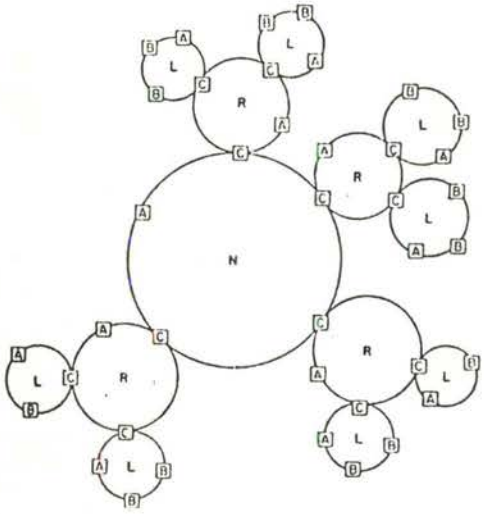


Fig. II.5 L'anneau de PIERCE.

Imaginons que dans la méthode de Farmer et Newhall on retire le message à destination et qu'on désire utiliser l'espace gagné sans attendre : les bytes seront émis sans problèmes à condition que le message nouvellement émis soit plus court que le message retiré, sinon il a un risque d'altérer le message qui suit.
Une solution serait alors de fixer la longueur des messages : c'est ce qui est fait dans l'anneau DCS, mais la capacité reste fort limitée parce qu'on ne retire tout de même pas le message à destination.

L'anneau de Pierce tend à accroître cette capacité en réutilisant la place libérée lors de l'extraction d'un message arrivé à destination.

Principe

Chaque boucle comprend un noeud de type A qui agit comme superviseur de la boucle, plusieurs noeuds de type B , les noeuds

qui communiquent, et un noeud de type C qui est la porte de communication entre deux boucles de niveaux différents.

Le noeud A agit comme une source qui génère initialement des segments (Slots) à intervalles réguliers. A chacun de ces segments, ou cases, est associé un drapeau (Flag) qui indique s'il est libre ou occupé par un message.
Les messages ont une longueur fixe et on s'arrangera pour obtenir des segments assez grands pour les contenir.

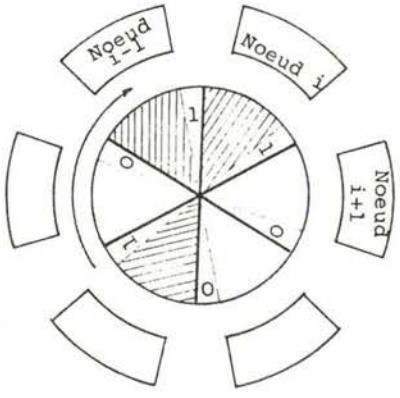


Fig. II.6 Technique du plateau tournant.

On a retenu l'idée de Farber en utilisant des messages de longueur fixe, mais en plus, dans ce système, plusieurs messages d'origines différentes peuvent circuler sur chaque boucle : cette technique est connue sous le nom de plateau tournant [CORNAFION].

En d'autres termes, c'est comme si les noeuds sont disposés autour d'un plateau qui tourne devant eux. Ce plateau, qui représente l'anneau, est divisé en un certain nombre de segments auxquels est associé un drapeau qui indique s'il est vide, c'est-à-dire qu'il ne contient aucun message, ou s'il est occupé par un message.

Travail des noeuds

Les noeuds A ne participent pas à l'échange de messages mais agissent seulement en temps que générateurs de segments libres ; ils se chargent aussi du contrôle de la boucle sur laquelle il sont situés et des messages perdus qui y circulent.

Un noeud B peut "déposer" son message sur la boucle dès qu'il rencontre un segment vide. En écrivant ses données dans ce segment, il modifie également le drapeau qui indique maintenant un segment plein.

Tous les noeuds B relaient le message jusqu'à destination. Là le message est copié et le drapeau est repositionné, de sorte que le segment redevienne libre.

Les noeuds C sont semblables aux noeuds B, mais ils s'occupent en outre d'aiguiller les messages adressés à des noeuds situés sur une autre boucle.

Format des messages

Les messages ont des formats de longueur fixe comme il a été dit. Seul leur entête varie suivant que le message ne doit circuler que sur une même boucle ou qu'il doit changer de boucle pour atteindre sa destination ; dans ce cas, l'entête contient toutes les informations nécessaires à son routage.

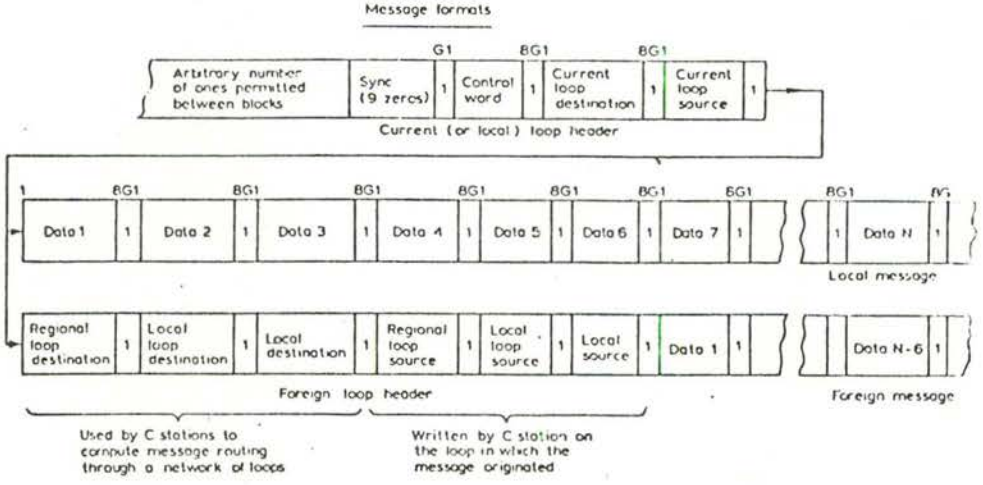


Fig. II.7 Format des messages de Pierce.

Remarques

1. L'architecture de Pierce ne prévoit pas de mécanisme d'acquittement, ce qui aurait d'ailleurs été complexe à implémenter vu la constitution hiérarchique de ce réseau. La probabilité que le message soit incorrectement reçu ou ne le soit pas du tout est donc toujours non nulle.

2. La stratégie utilisée (Réutilisation de la place libérée), permet d'accroître sensiblement la capacité du réseau puisque tout message est retiré de l'anneau à destination, et qu'il n'y a pas

de mécanisme d'acquittement.

3. Il existe toujours un noeud principal (Noeud de type A) qui supervise l'anneau. Il est toutefois étonnant de constater que ce noeud ne puisse également échanger de messages.

2.4. La "Spider Loop" de FRASER (1973)

(Technique du plateau tournant)

La boucle de Fraser est constituée d'un ensemble de boucles qui sont toutes contrôlées par un superviseur situé dans un même noeud central (Central Switch) : le contrôle du réseau est donc du type centralisé.

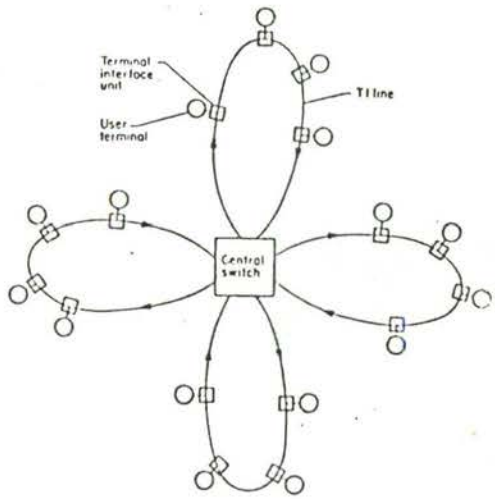


Fig. II.8a Boucle de FRASER.

Principe

La stratégie de fonctionnement de chaque boucle est la même que celle utilisée dans le réseau de Pierce : un certain nombre de segments, qui sont soit vides soit occupés suivant l'état du drapeau qui leur est associé, circulent en permanence sur la boucle. Les messages, qui y sont déposés, transitent d'abord de leur expéditeur vers le noeud central, même si les deux interlocuteurs sont situés sur la même boucle. Ce noeud central se charge ensuite de les aiguiller vers la boucle où se situe le noeud de destination.

Travail des noeuds

L'émetteur dépose, dès qu'il le peut, son message dans un segment qui circule sur la boucle sur laquelle il est situé. Le message arrive jusqu'au noeud central, même s'il rencontre en route son destinataire qui, d'ailleurs, ne le reconnaîtra pas. Ceci est dû au fait que les noeuds ne connaissent eux-même que leur adresse réelle, alors qu'ils sont adressés de manière virtuelle par les autres noeuds. Ce mécanisme a été adopté pour réduire le nombre d'informations nécessaire dans l'en-tête de chaque message.

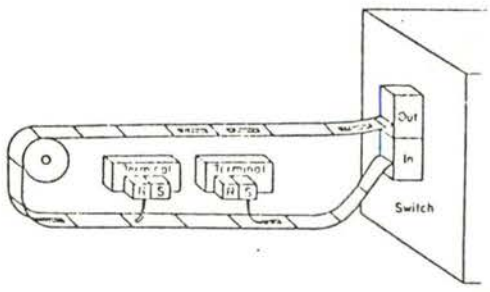


Fig. II.8b Une boucle du réseau de Fraser.

Le destinataire copie le message qu'il reconnait et le retire aussitôt de la boucle, permettant ainsi de réutiliser le segment qu'il occupait. Il n'y a donc plus de mécanisme d'acquittement dans cette architecture, comme d'ailleurs dans celle de Pierce.

Le noeud central se charge, lorsqu'un message y arrive, de convertir le nom virtuel du destinataire en une adresse réelle, puis il détermine la boucle sur laquelle il faudra envoyer le message pour qu'il arrive à destination.
C'est lui aussi qui génère les segments qui circulent sur les différentes boucles.

Format des messages

Les messages sont divisés en deux parties distinctes :

- L'entête du message : il comprend l'adresse, initialement virtuelle puis réelle, du destinataire, et l'adresse de l'expéditeur.

- Le message proprement dit, qui est lui-même décomposé en trois parties :

  - l'identification qui contient le numero de séquence du message, son type et sa longueur,

- les données elles-mêmes,

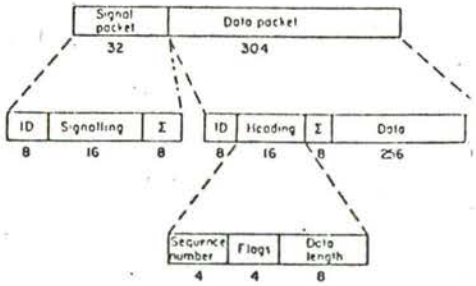


Fig. II.9 Format des messages de Fraser.

Remarques

Pour les transmissions qui concernent deux noeuds d'une même boucle, la technique perd tout son intérêt. Le message fait en moyenne un tour complet pour rejoindre la destination : un demi tour pour aller de l'expéditeur vers le noeud central, et un autre demi tour pour aller de celui-ci vers le destinataire.
Pour parer cet inconvénient, le principe suivant a été adopté : un noeud ne doit relayer un message que s'il ne lui est pas destiné. Ceci oblige les noeuds à connaître leur propre adresse virtuelle.

Du point de vue du fonctionnement isolé de chaque boucle, le réseau de Fraser se rapproche très fort de l'anneau de Pierce.

2.5. L'anneau de CAMBRIDGE (1975)

(Technique du plateau tournant)

L'université de Cambridge a développé en 1975 un réseau local, directement inspiré de celui de Pierce, pour supporter les applications de l'université.

Principe

Le réseau est composé d'un anneau sur lequel circulent continuellement des segments (Pierce : Technique du plateau tournant). Ces segments sont générés par un noeud moniteur, similaire au noeud de type A de Pierce, qui contrôle aussi les messages perdus.
Chaque segment possède son drapeau qui indique s'il est libre ou occupé.

Travail des noeuds

Un noeud qui veut émettre attend le premier segment libre qui passe devant lui et y dépose, outre les informations classiques de routage (Adresse d'origine, adresse de destination, Contrôle d'erreur, etc.) UN SEUL byte d'information utile. Ce noeud ne peut plus émettre d'autres messages tant qu'il n'a pas récupéré celui qu'il vient d'émettre. Ceci permet d'avoir une méthode d'acquittement, inexistante chez Pierce.

A sa destination, le message est copié mais n'est pas retiré de l'anneau : c'est son propre expéditeur qui s'en chargera lorsqu'il y sera de retour.

Remarques

1. Le protocole d'acquittement utilisé (Du genre de celui de Farmer) amène une perte réelle de 50 % de l'utilisation de l'anneau. Par ailleurs, il est évident, puisque chaque message est très court, qu'un acquittement n'est pas nécessaire pour chacun d'entre eux.

2. La rapidité de transmission dans cette architecture, près de dix fois celle de Pierce soit 10 Mbits/s, prime sur sa capacité.

2.6. L'anneau "DCLN" de LIU (1975)

(Technique du registre d'insertion)

Les architectures exposées jusqu'ici présentent toutes les mêmes inconvénients : présence d'un noeud principal, capacités plus ou moins restreintes du réseau suivant le mécanisme d'acquittement utilisé. L'inconvénient majeur réside dans la présence d'un noeud principal (Moniteur ou superviseur) qui a pour conséquence de centraliser le contrôle du réseau.

Dans cette optique, Liu a voulu développer un réseau d'ordinateurs au contrôle totalement distribué (Distributed Computer Loop Network) pour interconnecter entre-eux des ordinateurs de différentes tailles (Midi, mini et micro ordinateurs), des terminaux et d'autres périphériques, sans qu'aucun d'entre eux n'ait dans ses attributions une fonction de superviseur.

2.6.1. Critiques des architectures précédentes

Une critique des architectures qui ont été vues jusqu'ici peut être faite sur base de plusieurs critères :

2.6.1.1. Contrôle centralisé

Toutes les architectures précédentes sont pilotées par un noeud moniteur ou superviseur (Central switch de Fraser, noeud "A" de Pierce, etc.) ; ceci présente trois inconvénients majeurs :

1. La fiabilité de tout le réseau est menacée par la moindre panne d'un de ces noeuds superviseurs. Autrement dit, en terme de probabilité :

si Ps = la probabilité que le noeud superviseur tombe en panne et
si Pr = la probabilité que le réseau soit paralysé en raison d'une panne,
alors : Pr >= Ps.

2. Il faut développer, en général, deux types de noeuds, ce qui ne facilite certainement pas les tests et les expérimentations préliminaires.

3. AU niveau de la réalisation de ces réseaux, le fait de devoir initialement disposer d'un noeud superviseur va accroître les coûts de développement, d'autant plus que ces noeuds ne seront réalisés qu'en peu d'exemplaires.

2.6.1.2. Longueur des messages

Les messages qui circulent sur le réseau sont en fait des messages créés par des processus, au cours de leur exécution, à destination d'autres processus. La nature de ces messages dépend donc étroitement du traitement fait par ces processus (Transfert de fichiers, transfert de données binaires, de caractères, etc.)

Il semble donc, à priori, naturel d'utiliser des messages de longueur variable, voire "de longueur aussi grande que possible". Mais ceci soulève certains problèmes :

- Il faut prévoir des tampons de taille suffisante pour contenir ces messages,
- La probabilité qu'un bit d'information soit corrompu lors du transfert étant non nulle, la probabilité d'erreurs dans le transfert d'un message croît avec la longueur du message.

Il est donc nécessaire, par souci d'efficacité de limiter ou de fixer la longueur des messages.

La méthode de Farmer et Newhall permet de travailler avec des messages de longueur variable qui, après avoir quitté leur destination, occupent inutilement de la place. Il serait possible d'améliorer cette méthode en sachant à

l'avance que l'emplacement qui se présente au noeud peut-être utilisé.

Toutefois, la zone libérée étant de longueur inconnue puisque la longueur varie, il sera difficile pour un noeud de prendre la responsabilité de commencer l'émission d'un message dans cette zone, sans savoir s'il y aura assez de place pour l'y insérer sans qu'il en déborde, ce qui aurait pour conséquences d'écraser d'éventuels messages qui suivent.

On peut néanmoins envisager plusieurs solutions qui tenteraient d'éliminer ces limitations.

1ère solution : l'entête de chaque segment libre peut disposer, outre le flag d'occupation, d'un indicateur de longueur de la zone libérée. Cette méthode semble ne jamais avoir été réalisée. Elle nécessite une technique de "Garbage collector" pour rassembler les segments non utilisés.

2ème solution : puisque le problème provient de la variation de la longueur, il suffit de fixer cette longueur, soit L, pour supprimer le problème. Toutefois, le choix de la longueur reste délicat.

- si L est petit, la quantité d'information utile sera faible par rapport à la quantité d'information de service (Adressage, contrôle d'erreur) ; de plus, le pourcentage d'erreur étant lié à la longueur du message, le taux d'erreur sera faible et les acquittements seront rarement utiles.

- si L est grand, il y a un risque de perte de capacité, puisque des messages peuvent être plus courts que l'espace disponible dans chaque segment libéré ; le taux d'erreur et de retransmission seront plus élevés puisque les messages sont plus longs.

Ces remarques ne permettent pas d'affirmer que la méthode est mauvaise, mais soulignent simplement la difficulté du choix d'une longueur fixe.

2.6.2. L'anneau à insertion de registre

L'architecture proposée par Liu tente de combiner les avantages des méthodes précédentes sans en subir les inconvénients.
Ainsi, la méthode amène une solution attrayante au problème posé, en permettant l'utilisation de messages de longueur variable qui, originaires de noeuds différents, se partagent l'anneau.

Principe

Chaque noeud possède un registre à décalage au moins aussi grand que la longueur maximum (lmax) des messages qu'il peut envoyer. Ce registre sera utilisé dès que le noeud veut insérer un nouveau message sur l'anneau d'où l'architecture porte le nom d'insertion de registre.
Quand un noeud a un message à envoyer, il attend d'abord, s'il y a lieu, que le message qui circule devant lui soit complètement passé (Fig. II.8a).

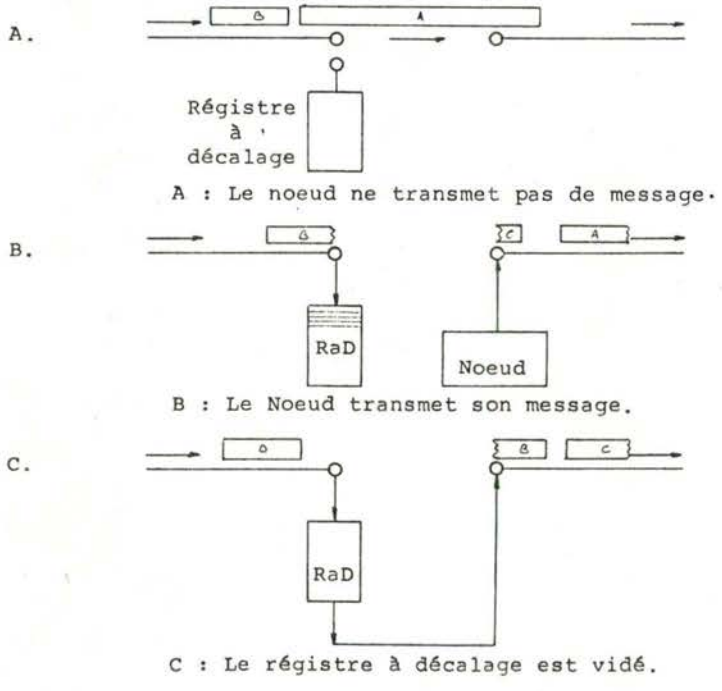


Fig. II.10 Le mécanisme de l'anneau DCLN.

Ensuite, il dérive l'entrée de l'anneau vers le registre à insertion, et dépose alors son message sur l'anneau (Fig. II.8b). Pendant l'émission de ce message (l<=lmax) d'autres messages reçus (l'<=l<=lmax) sont déposés dans le registre. Dès qu'il a terminé, la sortie du registre d'insertion est rebranchée sur l'anneau (Fig. II.8c), et le message retardé reprend son chemin. Le registre d'insertion sera occupé tant que des messages arriveront sur l'anneau. Au fur et à mesure que l'anneau n'envoie plus de messages, le registre d'insertion se vide, jusqu'à ce qu'il le soit complètement, pour être finalement déconnecté : il se comporte donc comme

une extension locale de l'anneau.

Le destinataire retire le message de l'anneau et envoie un acquittement, très court, vers l'expéditeur du message.

Format des messages

Le message "DCLN" (Fig. II.11) est composé des trois parties suivantes :

- L'en-tête qui comprend les adresses d'origine et de destination, permettant ainsi d'adresser un processus, et un ensemble d'informations relatives au type du message et au contrôle des messages perdus.

- La partie données.

- La queue qui comprend les informations nécessaires à la détection des erreurs de transmissions.

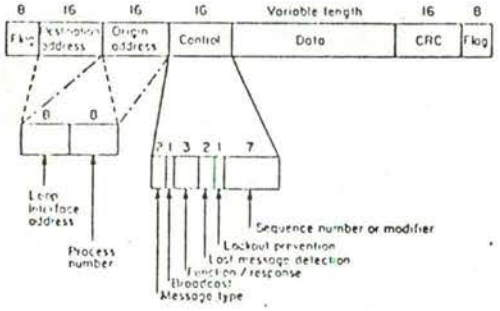


Fig. II.11 Format des messages DCLN.

Remarques

1. Le réseau DCLN répond aux problèmes posés :

   - L'utilisation de la capacité de l'anneau est optimale : il n'y a de "trous", ce qui correspond aux segments vides de la technique du plateau tournant, que si aucun noeud ne veut émettre de messages.

   - Il ne comporte pas de superviseur, le contrôle est donc entièrement distribué.

   - La longueur d'un message n'est pas seulement variable d'un message à un autre, mais peut aussi différer d'un noeud à un autre, suivant la longueur du registre d'insertion local.

- Aucun noeud ne peut se réserver l'utilisation de l'anneau comme dans le cas des techniques à jeton. Ici, un noeud émet dès qu'il le désire, et à concurrence de la taille de son registre d'insertion.

Les messages sont adressés à des processus situés dans des noeuds.

2. Il faut néanmoins prévoir des mécanismes pour la récupération des messages perdus, qui peuvent être supportés par chacun des noeuds [LOBELLE 2].

3. La mise hors circuit d'un seul noeud (Panne, défaillance) paralyse tout le réseau si on ne prévoit pas un mécanisme qui court-circuite le noeud (Relais).

Chapitre 3

3. RESEAUX BUS

Avant la fin des années 60, les réseaux bus se limitaient pratiquement aux lignes multipoints ou bus de communication internes des ordinateurs. Ces structures sont caractérisées par le fait de l'accès déterministe : avant d'obtenir ce bus et de l'utiliser comme moyen de communication, les noeuds, qui y sont connectés, doivent obtenir préalablement l'autorisation d'accès soit d'un organe maître, soit par concertation entre noeuds grâce à un moyen spécialisé. L'accès, subordonné à cette autorisation, est régit selon une séquence répondant à des règles prédéterminées.

Depuis lors, la plupart des réseaux bus utilisent une méthode d'accès aléatoire qui n'exige pas d'autorisation préalable pour émettre ; l'émision devient l'initiative propre de chaque noeud.

Le développement des réseaux du type bus a eu un précurseur en la matière ; ce précurseur n'est pas à proprement parlé un réseau local puisqu'il s'étend sur plusieurs centaines de kilomètres ...

3.1. Un précurseur : le réseau ALOHA (1969)

(Technique de l'accès aléatoire)

C'est en 1969 que le réseau Aloha est entré en opération. Il était révolutionnaire quant à sa conception sur deux points au moins :

- L'originalité dans l'utilisation des voies hertziennes comme moyen de communication plutôt qu'un réseau câblé.
- Le fait de ne plus utiliser une méthode d'accès déterministe.

Ce réseau a été conçu par l'université d'Hawai, et devait relier un ordinateur situé à Honolulu avec plusieurs terminaux intelligents dispersés dans quelques îles du Pacifique. L'ordinateur émet vers chaque terminal par le biais d'un canal radio de diffusion. Le canal de réception de l'ordinateur étant unique, les terminaux désirant émettre vers l'ordinateur doivent se partager ce canal commun.
Ce partage se fait donc de manière aléatoire sans autorisation préalable.

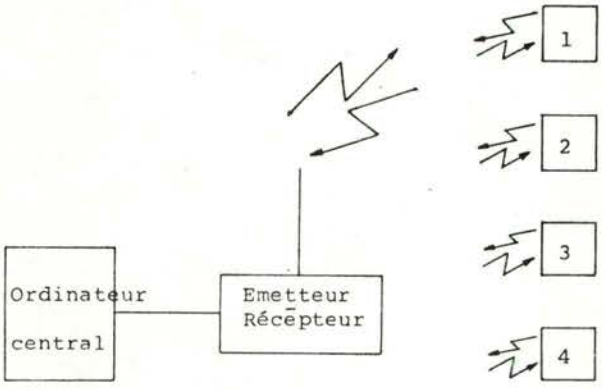


Fig. II.12 Le réseau ALOHA.

3.2. Méthode d'accès aléatoire utilisée dans le réseau Aloha

Les terminaux n'ont pas d'accord entre eux pour déterminer celui qui peut utiliser le canal le premier. Quand un d'entre eux a un message à transmettre à l'ordinateur, il l'envoie sans se préoccuper des autres.

Si le terminal a de la chance, aucun autre n'émet à ce moment là, son message arrive correctement à l'ordinateur. Celui-ci renvoie alors un acquittement au terminal.

Si par malchance, il émet juste quand un autre terminal émet son message, les deux messages s'embrouillent l'un l'autre. L'ordinateur d'Honolulu ne reconnait aucun message valide, et n'envoie donc aucun acquittement. Après un certain laps de temps, les terminaux entrés en collision décident que leurs messages sont perdus et qu'ils doivent réessayer.
S'ils le font après un laps de temps fixe, ils rentreraient à nouveau inévitablement en collision. C'est pourquoi ils attendent chacun pendant un laps de temps aléatoire avant de réessayer, jusqu'à l'aboutissement d'une émission correcte.

3.3. Amélioration de la méthode

La méthode exposée, l'accès aléatoire, est cependant trop simpliste et le besoin d'améliorer le système s'est rapidement fait sentir : il est stupide de voir un message corrompu par l'émission d'un autre terminal, alors que le début de sa transmission est parfait. De même, il n'est plus utile de continuer d'émettre la suite d'un message corrompu puisqu'aucun acquittement n'y fera suite.

Une première amélioration prévoit que chaque terminal écoute avant d'émettre (Listen before talk) de manière à vérifier qu'aucun autre message ne soit actuellement en cours de transmission.
Ceci évite, en principe, de faire rater une émission qui avait bien commencé, mais ne garantit tout de même pas qu'il n'y aura plus de collision : deux terminaux, ayant chacun trouvé le canal libre, peuvent commencer simultanément la transmission de leurs messages qui rentreront en collision.

Ceci nous conduit à la seconde amélioration faite à la méthode d'Aloha : le terminal écoute son message pendant qu'il l'émet (Listen while talking). Il pourra ainsi détecter si ce message a été corrompu par un autre message. Si c'est le cas, il stoppe immédiatement son émission, il n'y a en effet plus de raison de continuer, et il attend un laps de temps de durée aléatoire avant de réessayer.

3.4. Le réseau ETHERNET (1975)

(Technique du bus à contention)

Le réseau Ethernet s'inspire directement du principe d'Aloha, d'où il tire d'ailleurs son nom : l'Ether (En anglais), croyait-on au XIXè siècle, est un gaz omniprésent qui porte les signaux radio.

L'idée de base d'Ethernet est de trouver un support autre que les voies hertziennes, qui permette le même type de stratégie d'accès dans un seul bâtiment. Ce support, appelé bus passif, est un câble coaxial auquel sont raccordés tous les noeuds.

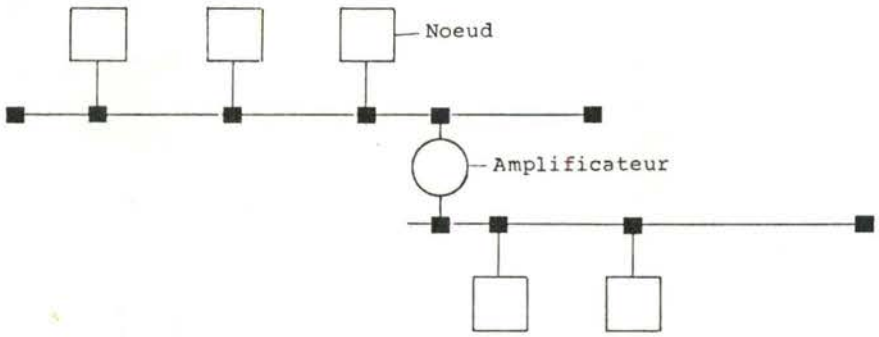


Fig. II.13 Le réseau ETHERNET.

Principe

Lorsqu'un noeud veut émettre, il écoute le bus, et attend qu'il soit libre, c'est-à-dire qu'aucun autre message ne soit en cours de transfert. Lorsque cette condition est vérifiée, le noeud peut commencer son émission.
Simultanément, il écoute le bus pour vérifier que ce qui est transmis correspond à ce qu'il a émis.

Le cas de discordance correspond à une collision, où deux noeuds, ou plus, ont émis leur message en même temps. Dans ce cas, chaque émetteur abandonne le transfert en cours et ne recommencera l'émission du message qu'au bout d'une durée de temps fixée aléatoirement, ceci pour diminuer la probabilité d'une seconde collision entre les mêmes messages.

Par ailleurs, tous les noeuds sont constamment à l'écoute du bus, et prennent sur celui-ci tous les messages qui leur sont adressés.

Format des messages

Le format des messages a été étudié de sorte qu'un noeud sache au plus vite si le message qui circule sur le bus lui est destiné. L'adresse du destinataire est pour celà disposée en tête du message.

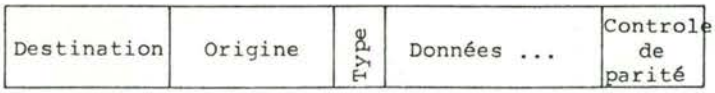


Fig. II.14 Format des messages d'Ethernet.

Remarques

1. Le principe utilisé est souvent frappé du sigle CSMA/CD pour Carrier Sense, Multiple Access, Collision Detection, ce qui veut dire :

   - Carrier Sense : détection de la présence d'un signal, et par extension d'un message, sur le bus.

   - Multiple Access : la ressource "bus" est une ressource partagée par tous les noeuds qui y font des accès concurrentiels.

   - Collision detect : détection de l'occurrence d'une collision.

   On l'appelle aussi souvent Bus à contention.

2. Il est important de noter qu'une collision ne peut plus survenir que dans le début de l'émission du message. Supposons que le noeud i, situé à une extrémité du bus, vienne de commencer l'émission de son message. Le signal n'a pas encore eu le temps d'atteindre l'autre extrémité du bus où se trouve un noeud j. Ce dernier peut encore trouver le bus libre, grâce au carrier sense, bien qu'une émission soit déjà en cours, et peut aussi commencer à émettre.
   Par contre, si le premier caractère a eu le temps d'atteindre le noeud j, au moment ou celui-ci désire émettre, il trouve le bus occupé, et doit attendre, avant d'émettre, que le bus redevienne libre.

   Une collision ne peut donc survenir, au pire, que pendant le temps de propagation du premier caractère à travers tout le bus.

3. Actuellement, on tente de développer l'utilisation de câbles coaxiaux à large bande, partagés entre plusieurs services de

communication (Voix et radio, vidéo et TV, données, etc.) grâce à un multiplexage de fréquences (FDM). Citons comme exemple, le réseau Mitrenet.

4. La stratégie d'accès aléatoire, souvent mentionnée comme celle d'Ethernet, devient de plus en plus un standard en matière de réseau bus.

TROISIEME PARTIE

CONCEPTION DES RESEAUX LOCAUX

INTRODUCTION

Le besoin généralisé de pouvoir interconnecter des équipements informatiques de différents constructeurs formant ainsi des réseaux hétérogènes, a conduit l'Organisation Internationale de Normalisation (ISO) à mettre en place dès 1977 un nouveau comité (SC-16) qui s'intéresse à l'"interconnexion de systèmes ouverts" [1].

Dès le démarrage de ses travaux au début 1978, le SC-16 se consacre en priorité à la définition d'une norme d'architecture de réseau informatique au sein de laquelle puissent prendre place les protocoles normalisés qui permettront la constitution de réseaux hétérogènes.

Le but ne sera pas d'exposer ici tous les travaux du SC-16, mais plutôt d'essayer de voir comment ces travaux peuvent s'adapter au cas particulier des réseaux locaux.

---

1 : Voir [ANSI], [DESJARDINS] et [JACOBSEN]

Chapitre 1

1. LE MODELE DE REFERENCE POUR L'INTERCONNEXION DE SYSTEMES OUVERTS (OSI)

1.1. Préliminaires

OSI fait référence aux standards d'échange d'information entre les terminaux, les ordinateurs, les réseaux, les processus et mêmes les personnes, qui sont ouverts les uns aux autres. Le terme ouvert n'exige pas quelqu'implémentation particulière, ni des moyens de connexion ou des technologies, mais il fait référence au principe de reconnaissance mutuelle et de support de standards.

Le modèle de référence d'OSI (RM/OSI) fournit une base commune pour la coordination de développements des standards de chacune des sept couches qui le composent. Il permet aussi l'évaluation et l'amélioration des standards existants avec comme objectif de rencontrer les besoins futurs des systèmes distribués qui doivent interagir entre eux pour exécuter des tâches communes (distribuées).

1.2. Concepts de base

Un système est un ensemble d'ordinateurs, de périphériques, de terminaux, de logiciels, de moyens de communication et d'opérateurs humains qui forment une entité autonome capable de traiter de l'information.

Un processus d'application (PA) est l'élément d'un système qui exécute un traitement d'information pour une application déterminée.

Le transport d'information entre les systèmes, est réalisé grâce à un moyen physique d'interconnexion de systèmes.

OSI veut permettre à un PA, situé dans un système ouvert qui supporte les standards OSI, de communiquer avec n'importe quel autre PA, situé dans n'importe quel autre système, auquel le premier est relié via des moyens de communication. Comme cas particulier, deux PA peuvent résider dans le même système, mais ils n'ont normalement pas connaissance de ce fait.

Lorsque deux PA veulent communiquer, une association logique s'établit entre eux, au travers de réseau de communication qui leur permet d'échanger de l'information en respectant les protocoles d'OSI.

Les objectifs d'OSI sont donc d'obtenir des échanges d'information fiables entre n'importe quel PA de n'importe quel système ouvert situé n'importe où sur la Terre.

1.3. Découpe en couches

L'architecture adoptée pour RM/OSI fait intervenir le concept familier de couches. Par cette approche, une structure très complexe est divisée en un certain nombre de couches fonctionnelles indépendantes.

Chaque couche réalise un ensemble bien défini de fonctions, en utilisant un autre ensemble bien défini de fonctions fournies par la couche juste inférieure.
Ces fonctions forment un ensemble de services qui peuvent être demandés par la couche supérieure. La communication entre les différentes entités qui appartiennent à une même couche est réalisée par des protocoles de la couche, sur base de connexions fournies comme services par la couche inférieure.

1.3.1. Principes de décomposition en couches

Il est difficile de prouver que telle ou telle architecture est la meilleure solution possible. Toutefois, il existe des principes généraux qui peuvent aider à trouver les limites des couches.
Ces principes sont :

P1. Ne pas créer trop de couches pour ne pas rendre trop difficile la conception du système, la description des tâches et l'intégration des couches.

P2. Etablir des limites là où la description des services peut être simple et où le nombre d'interactions à travers ces limites est minimum.

P3. Créer des couches séparées pour des fonctions manifestement différentes.

P4. Rassembler les fonctions similaires dans une même couche.

P5. Etablir les frontières là où l'expérience antérieure avait déjà fait ses preuves.

P6. Créer des couches de fonctions facilement localisables, de telle sorte qu'une couche puisse être entièrement remaniée

sans avoir d'incidence ni sur les services qu'elle procure, ni sur les interfaces qui la relie aux autres couches.

P7. Etablir une frontière là où il peut être utile plus tard d'avoir un interface standardisé.

P8. Créer des couches pour différents niveaux d'abstraction à manipuler.

P9. Veiller à permettre des changements de fonctions ou de protocoles dans une même couche sans affecter pour autant les autres couches.

P10. Créer des interfaces en correspondance avec les deux couches qu'ils relient.

P11. Veiller à distinguer des sous-couches dans une même couche, au cas où des services de communication particuliers seraient nécessaires.

P12. Autoriser le court-circuitage d'une couche.

1.3.2. Les septs couches d'OSI

Ainsi, sur base des principes qu'on vient d'énoncer, RM/OSI définit une architecture standard en sept couches :

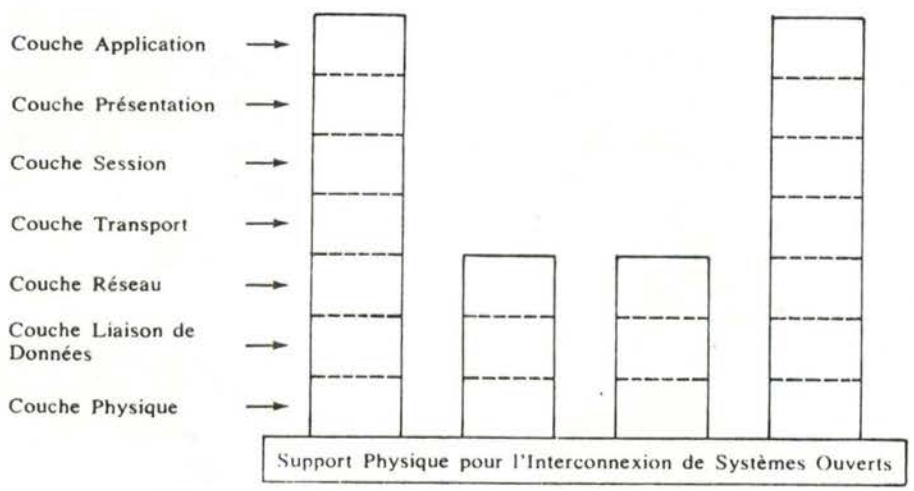


Fig. III.1 Modèle de référence pour l'interconnexion des systèmes ouverts.

1. La couche application comprend les programmes d'application avec leurs conventions d'échanges et de coopérations. On y retrouvera donc des protocoles spécifiques selon les types d'applications (Applications bancaires, réservations de

places, etc.). Ces services profitent directement à l'utilisateur de l'environnement d'OSI.

2. La couche présentation est responsable de la présentation des données échangées par les applications de manière à résoudre les différences syntaxiques ou sémantiques, tout en gardant le sens de l'information (Exemple : entre un serveur et un terminal).

3. La couche session fournit les moyens nécessaires aux couches de présentation qui coopèrent, pour organiser et synchroniser leurs dialogues, et contrôler leurs échanges de données.

4. La couche transport est responsable du contrôle du transport d'information de bout en bout, entre deux processus, au travers d'un réseau, indépendamment des moyens utilisés. Elle doit aussi optimiser l'utilisation des ressources disponibles pour obtenir les performances, exigées par ses utilisateurs, à un coût optimal.
   Ses utilisateurs sont connus par une adresse.

5. La couche réseau permet d'établir, de maintenir et de supprimer des connexions entre systèmes (Endpoints) qui comportent des applications coopérantes. C'est elle qui s'occupe de tous les problèmes de routage et d'aiguillage associés à une connexion-réseau.
   C'est ici qu'on trouvera, par exemple, le niveau paquet du protocole X25.

6. La couche liaison de données est responsable de l'acheminement, sans erreurs, de blocs d'information sur des liaisons de données (Point to point), pour l'établissement et la désactivation desquelles elle fournit des moyens fonctionnels et procéduraux.

7. La couche physique comprend les moyens mécaniques, électriques, fonctionnels et procéduraux pour activer, maintenir et désactiver les connexions physiques utilisées pour la transmission de bits d'information.

Chapitre 2

2. APPLICATION DU MODELE DE REFERENCE AUX RESEAUX LOCAUX

Un réseau local n'a pas besoin de retenir toutes les couches qui viennent d'être décrites. Seules celles qui sont nécessaires, entreront en ligne de compte [Principe P12].

2.1. Les couches retenues

Si les standards OSI sont plus ou moins suivis en matière de réseau longue distance, il n'en est pas de même en ce qui concerne les réseaux locaux comme on pourra le voir dans la suite.
Les couches telles qu'on les trouve, dépendent encore très fortement du modèle de topologie utilisé (Anneau ou bus).

1. On retrouve en premier lieu la couche application qui comprend de manière générale les processus d'application qui communiquent. Ces processus réalisent eux même les fonctions de session et/ou de présentation, quand c'est nécessaire. La couche application reprend donc les trois couches supérieures de l'architecture OSI.

2. La deuxième couche est la couche transport, qui, indépendamment du moyen de communication utilisé, fournit un service de transport de bout en bout entre deux processus.
   La couche s'occupe également du contrôle de séquence, de la détection des erreurs sur ces séquences, de la manipulation des adresses source et destination, ainsi que de la délimitation des trames.

3. Vient ensuite la couche liaison de données qui se charge des transmissions point à point. Elle a dans ses attributions les fonctions de synchronisation et de délimitation, qui permettent de reconnaître des séquences de bits transmises sur la connexion physique, une fonction de détection d'erreur, dues soit à la connexion physique, soit à un mauvais fonctionnement de la couche elle-même.

4. Enfin, la couche physique qui fournit les services de connexion point à point utilisés par la couche de liaison de données et délivre les bits d'information dans le même ordre que celui dans lequel ils ont été transmis.
   Elle indique, le cas échéant, les erreurs détectées à la couche supérieure.

2.2. Application à trois architectures de réseaux

Bien que des standards du type OSI soient en cours de développement en ce qui concerne les réseaux locaux [ECMA], il est intéressant de voir comment des architectures réelles s'inspirent d'une découpe en couches.

2.2.1. Présentation des trois architectures

Les architectures comparées sont :

- Ethernet de Xerox (Bus),
- Monet de l' ULg (Bus),
- Trout de l' UCL (Anneau).

<table>
<tr><th>O S I</th><th>ETHERNET</th><th>MONET</th><th>TROUT</th></tr>
<tr><td>APPLICATION</td><td rowspan="2">CLIENT</td><td>HAUT - NIVEAU</td><td>APPLICATION</td></tr>
<tr><td>TRANSPORT</td><td>TRANSPORT
(Endpoints)</td><td>TRANSPORT
(Endpoints)</td></tr>
<tr><td>LIAISON DE DONNEES</td><td>DATA LINK</td><td rowspan="2">(Collision Detect)
(Control. de jonction)
TRANSMISSION
(Jonction)
(Câble)</td><td>(Node to Node)
TRANSMISSION
(Hardware)</td></tr>
<tr><td>PHYSIQUE</td><td>PHYSICAL</td></tr>
</table>

Fig. III.2 Tableau comparatif de trois architectures.

La figure III.2 donne un tableau comparatif de ces trois réseaux

et de leurs architectures en couches.

2.2.2. Comparaison des trois architectures

Dans chacune des architectures qu'on s'est proposé de comparer, on ne retrouve plus que trois couches.
De manière générale, on remarque cependant que dans chaque cas, une des trois couches est subdivisée, du point de vue fonctionnel, en deux sous-couches, nous permettant ainsi d'obtenir quatres couches et sous-couches qui rencontrent celles du modèle proposé.

Voici pour les trois architectures précitées, la description des couches qui les composent, en partant du niveau supérieur vers le niveau inférieur ; on a gardé l'appelation des couches telle qu'elle apparaît dans les spécifications de chacune des architectures.

1. Ethernet [DIGITAL]

Son architecture s'inspire directement de RM/OSI de sorte que :

a. Les deux couches supérieures du modèle proposé (Application & transport), ne faisant pas partie des spécifications d'Ethernet, sont toutes rassemblées en une seule couche appelée couche client.

b. La couche liaison de données se charge des fonctions de délimitation des paquets envoyés (Encadrage par des caractères de contrôle), de gestion de la liaison (Prévention et gestion des collisions, synchronisation) et de gestion des erreurs qui pourraient survenir.

c. La couche physique s'occupe du traitement des bits d'information : encodage et désencodage, détection d'une porteuse (Carrier sense), transmission et réception de bits.

2. Monet [DANTHINE]

Le réseau local Monet de l'Université de Liège est principalement un réseau bus du style d'Ethernet, destiné à interconnecter des terminaux et un ordinateur. Il est intéressant de remarquer les différences dans la structure architecturale :

a. La couche de haut niveau comprend les processus d'application qui ne sont dans ce cas-ci que des terminaux (A comparer avec les mini-ordinateurs Alto connectés au réseau initial Ethernet). Ces terminaux étant généralement peu sophistiqués, on

ajoute des prèprocesseurs qui gèrent leurs accès au rèseau.

b. la couche transport de Monet correspond intègralement à la couche transport de RM/OSI; elle fournit deux types de services :

   - adressage des processus au niveau du rèseau ;
   - communication en mode liaison basée sur l'établissement d'une liaison virtuelle.

c. la couche transmission reprend d'une part les fonctions rèalisèes par la couche liaison de RM/OSI (Encapsulage des paquets, gestion des collisions et des erreurs) et d'autre part, celles rèalisèes au niveau physique (Encodage, dètection de porteuses, transmission et rèception de bits).

3. Trout [LOBELLE 2]

Le rèseau trout de l'UCL est un rèseau à insertion de règistre du type de celui de Liu (DCLN). On y retrouve une architecture fort semblable à celle qu'on vient de dècrire pour Monet. On citera donc pour rappel :

a. La couche application qui comprend les processus d'application.

b. La couche transport qui fournit les moyens procèduraux pour assurer un service de transport de bout en bout.

c. La couche transmission qui, au niveau d'une liaison de donnèes, fournit le service de transport point à point (Node to node) et, au niveau physique, le service de transmission de bits d'information.
Cependant, on y trouve deux services supplèmentaires dus à la topologie "Anneau" du rèseau, qui se chargent du retrait des messages perdus et du contrôle du règistre d'insertion.

2.2.3. Conclusion

Le but n'est pas ici de faire une critique des architectures proposèes, mais d'en tirer profit au niveau des difficultès d'établissement de standards.
On trouve, en effet, deux architectures diffèrentes pour deux rèseaux à technique similaire (Ethernet et Monet), et deux architectures similaires pour deux topologies diffèrentes (Monet et Trout).

2.3. Les interfaces réseaux

A l'exception de la couche application, les trois couches inférieures de notre modèle (Transport, liaison de donnée et physique) peuvent être regroupées et fournissent dès lors à l'utilisateur, c'est-à-dire aux processus d'application, le service global d'accès au réseau ; l'ensemble des fonctions, qui réalisent ce service est appelé interface réseau (Network Interface Unit : NIU).

2.3.1. Utilités des interfaces réseaux

Indépendamment d'une architecture en couches, les fonctions principales réalisées par un NIU sont :

1. Le contrôle de séquence : les messages doivent être délivrés dans leur ordre d'émission.

2. La copie des messages qui sont destinés aux PA qui dépendent du NIU et l'envoi de ces messages au PA correspondant.

3. La retransmission des messages destinés à d'autres NIU, si ce n'est pas un réseau bus.

4. La validation des messages, détection des erreurs et autres actions appropriées (court-circuitage des noeuds défectueux dans le cas d'un réseau en anneau).

5. L'assemblage des données pour former un bloc de données ; l'encapsulage de ce bloc par un en-tête et une queue qui comprennent respectivement les adresses d'origine et de destination, les informations nécessaires au protocole et au contrôle d'erreur.
   Un travail inverse, le décapsulage et le désassemblage, est fait à la réception.

2.3.2. Services fournis par un interface réseau

Grâce au support des fonctions qui viennent d'être définies, un interface réseau fournit deux services globaux aux processus d'application. Ces services sont :

- L'envoi de message d'un PA à n'importe quel autre PA du réseau.

- La réception de message en provenance de n'importe quel PA du réseau.

2.3.3. Implémentation des interfaces réseaux

Jusqu'ici, on ne s'est penché que sur les principes de décomposition en couches des réseaux de systèmes ouverts (OSI).

Les interfaces réseaux constituent par eux-mêmes des équipements spécialisés qui fournissent aux PA un service global (P. ex : liaison de données dans Ethernet). De cette sorte, ils sont aussi souvent distincts des équipements auxquels ils servent, et apparaissent dès lors plus comme une porte d'accès (Gateway) à un réseau.

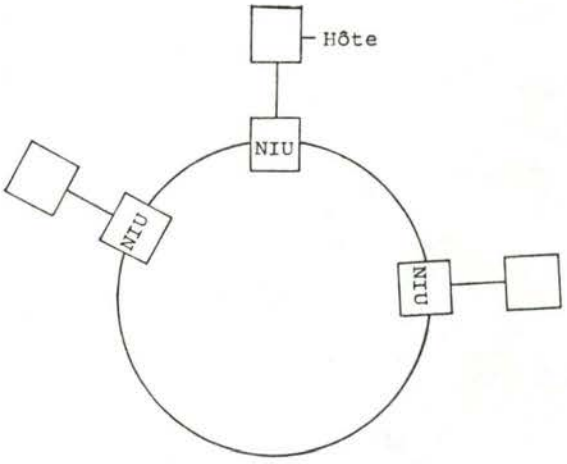


Fig. III.3 Les NIU : portes d'accès au réseau.

Mais, ils peuvent aussi très bien faire partie intégrante de l'appareil à connecter ; ceci sera principalement vrai en ce qui concerne tout ou partie du logiciel de communication.

Les interfaces réseaux comportent en fait deux catégories distinctes de fonctions :

- les fonctions physiques qui réalisent essentiellement les fonctions de transformation d'information logique en signaux électriques, et inversement ;

- les fonctions logiques ou procédurales, qui réalisent les fonctions du protocole de communication proprement dit.

Les fonctions physiques sont réalisées sous forme de composant hardware. Les fonctions procédurales peuvent être implémentées soit sous forme de logiciel, en software voire firmware, soit sous forme de composant hardware ; le choix de telle ou telle implémentation dépend bien évidemment des caractéristiques et des spécifications du NIU.

On peut citer comme exemple Ethernet, où le NIU réalise lui-même la fonction liaison de données et celles du niveau inférieure nécessaires au bon fonctionnement de celle-ci.

Les fonctions de transport ou de niveaux supérieurs sont à réaliser par l'ordinateur connecté au NIU.

De même, dans Hinet de DMS, le NIU se charge de la résolution des accès physiques au réseau mais tout le protocole de communication, qui lui est propre, depuis la couche liaison de données jusqu'au PA est à implémenter dans l'ordinateur-même.

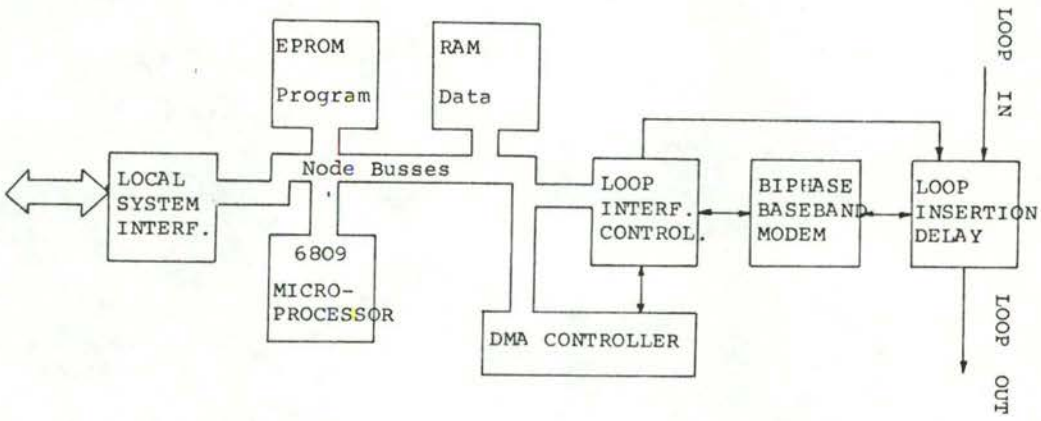


Fig. III.4 L'interface Trout.

Dans le réseau Trout, le NIU devient alors vraiment un interface intelligent, piloté par un microprocesseur, qui réalise lui-même tout le protocole d'accès au réseau.

QUATRIEME PARTIE

MISE EN OEUVRE D'UN PROTOCOLE DE COMMUNICATION

Chapitre 1

1. L'ENVIRONNEMENT DE TRAVAIL

Dans le cadre du développement d'un ordinateur à stockage de masse (G-MASS) pour le laboratoire de l'université de Lille, il est apparu intéressant de connecter, en outre, celui-ci à l'ordinateur central de l'université (IRIS 80 de CII-HB).

1.1. Présentation de l'environnement

G-MASS est une machine originale permettant à d'autres sites (1), principalement des micro-ordinateurs (systèmes de développement INTEL) et d'autres projets du laboratoire, de partager une ressource commune, à savoir, une mémoire secondaire (en fait deux disques durs de 20 M-octets chacun). Le partage de cette ressource ne se limite pas au stockage de masse mais peut également déboucher sur d'autres utilisations comme la communication inter-processus à l'aide de boites aux lettres.

1.2. Configuration de G-MASS

G-MASS est un ordinateur piloté par un microprocesseur maître 8085 qui (Fig. IV.1)

- dirige les différents microprocesseurs esclaves, également des 8085, qui s'occupent du dialogue avec les sites connectés;
- gère les accès à la ressource commune via un controleur disque dans un environnement temps réel (RMX-80 de INTEL).

L'interconnection des sites du laboratoire avec G-MASS est réalisée de manière très particulière et ne sera pas exposée ici puisque nous sommes principalement intéressés par la connexion de G-MASS à l'ordinateur central de l'université.

---

(1) Site : tout processeur d'information, c'est-à-dire ordinateurs, micro-ordinateurs, terminaux, périphériques, etc.

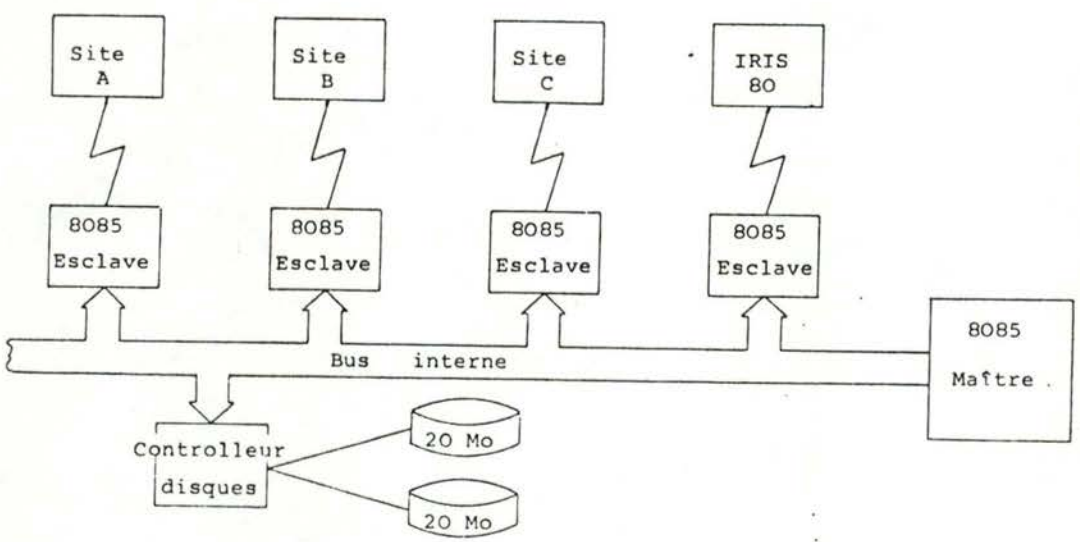


Fig. IV.1 G-MASS

Chapitre 2

2. L'INTERCONNEXION G-MASS - IRIS-80

Comme dans tout environnement de téléinformatique, il faut aborder les deux aspects, hardware et software, de la connexion.

Au niveau hardware nous parlerons des interfaces, modems et lignes de transmission, tandis que les protocoles de communication feront partie du niveau software.

Fig. IV.2 Interconnexion INTEL 8080 - IRIS 80.

2.1. Configuration hardware

L'étude et le développement de cette liaison ont été réalisés non pas sur G-MASS, puisque cette machine est toujours en construction, mais sur un système de développement (INTEL 8080 Microprocessor); ce micro-ordinateur est piloté par un microprocesseur 8080 (compatible avec le microprocesseur 8085 mais de capacités inférieures) et utilise le même système d'exploitation en temps réel (RMX-80) qui équipera la nouvelle machine.

On peut schématiser l'interconnexion qui a été établie entre les deux sites pour réaliser le travail (Fig. IV.2).

2.1.1. La ligne de transmission

La ligne de transmission qui relie les deux sites est constituée d'un câble à deux fils, un fil de donnée (Data) et un fil de masse (Ground). Cette ligne est connectée à un bout à l'IRIS-80 et à l'autre, au système de développement d'INTEL; c'est cette extrémité-là qui nous intéressera. En effet, la connexion à l'IRIS-80 existe déjà et est utilisée conjointement avec d'autres utilisateurs sous forme de ligne dite ligne multipoints.

2.1.2. Connexion au système de développement INTEL 8080

La connexion de la ligne de transmission au système de développement se fait grâce à une carte interface ISBC 544 d'INTEL [2].

Il s'agit d'un interface intelligent, i.e. programmable, composé de plusieurs circuits de contrôle dont les trois suivants retiennent principalement notre attention :

- Un interface série de communication programmable INTEL 8251 ;
- Un controleur d'horloge INTEL 8253 (Programmable Interval Timer ou PIT) ;
- Un controleur d'interruption INTEL 8259 (Programmable Interrupt controller ou PIC).

Une description succinte de ces circuits et de leurs fonctions figure en annexe [Annexe D2].

2.2. Configuration software

Bien sûr, une autre partie , tout aussi considérable dans un problème de communication, se situe au niveau software où sera réalisé le protocole qui comprend les conventions nécessaires pour faire cóperer les différents correspondants entre eux.

Avant d'aborder la définition précise du protocole considéré, il est à noter que plusieurs contraintes peuvent peser sur le projet. Dans notre cas, il s'agissait principalement de

1. Pouvoir travailler en temps réel et de rendre la communication aussi transparente que possible, ce qui exige une certaine rapidité du logiciel de contrôle du protocole;

2. Réduire la longueur du code objet de ce logiciel pour pouvoir le geler par la suite sur mémoire morte (EPROM).

On comprend aisément la nécessité de pareilles considérations si on se souvient de l'objectif premier du projet G-MASS qui doit gérer une ressource commune à plusieurs processeurs.

Chapitre 3

3. LES PROCEDURES DE TRANSMISSION

3.1. La famille de protocoles TMM

Le choix de la procédure TMM n'est pas fortuit puisqu'il s'agit là d'une famille de protocoles développés par CII-HB et qui est donc intégrée dans les ordinateurs du type IRIS.

La famille TMM (Transmission en Mode Messages) comprend principalement les procédures suivantes :

- TMM-VU : pour console de visualisation et concentrateur-diffuseur de messages;
- TMM-RB : (Remote Batch) pour terminal de traitement par lots à distance;
- TMM-UC : pour liaison entre deux ordinateurs.

3.2. Le mode de base

Les protocoles TMM sont des protocoles dits "Mode de base" dont les fonctions essentielles sont définies par la norme ISO-1745-75 (1). Les principales fonctions du mode de base sont décrites dans les annexes [Annexe B1] ; le lecteur s'y référera pour compléter son information.

---

(1) La norme Mode de Base est également définie dans :
AFNOR-NF-Z-66-010, 011, 015-020 ,
ISO 2111, 2628, 2629 ,
ECMA 16, 24, 27-29 .
Voir aussi [MACCHI]

3.3. Le protocole TMM-RB

3.3.1. Spécification des fonctions utilisées

La transmission s'effectuera en série, mode synchrone d'octets, bits de poids faibles en tête. Tout message est précédé et suivi d'une séquence de caractère de synchronisation SYN dont le nombre est ajustable, et terminé par un caractère spécial PAD marquant la fin du message. La longueur de celui-ci ne peut pas dépasser 256 caractères.
Un contrôle de parité impaire est élaboré transversalement sur tous les caractères et longitudinalement sur les blocs d'information seulement. Dans la suite du texte, l'extrémité qui a l'initiative des échanges sera appelée station centrale ou le central; l'autre, qui se contente de répondre, station terminale, c'est la station.

3.3.1.1. Types de périphériques

A ce niveau-ci, la station peut être vue du central comme un regroupement de périphériques divers; on y retrouve entre autre:

- une console utilisateur,
- un lecteur-perforateur de cartes,
- une imprimante,
- un périphérique "fictif".

Le premier de ceux-ci, la console utilisateur, est le périphérique normalement choisi pour la communication ou du moins, le plus fréquemment.

Les deux suivants, le lecteur-perforateur de cartes et l'imprimante, sont choisis plus rarement, le premier des deux en polling (en temps que lecteur de cartes) comme en sélection (en temps que perforateur de cartes) et le second uniquement en sélection. La possibilité d'utilisation de ces deux périphériques est subordonnée à une autorisation; cette autorisation est fournie par le programme d'application lui-même qui positionne une variable à une certaine valeur.

Enfin, le dernier périphérique : il s'agit non pas d'un périphérique matériel, mais bien, comme son nom l'indique, d'un périphérique "fictif" dont le rôle est de permettre le test de la ligne dans les deux sens; en scrutation, les blocs émis seront des blocs "fictifs" dont le contenu n'est pas nécessairement connu; en sélection, les blocs reçus ne sont pas traités puisque l'information qu'ils contiennent n'est pas utile. C'est le périphérique fictif qui gère alors tant les blocs émis que ceux reçus.

3.3.1.2. Types de messages

Comme on l'a déjà vu, l'ensemble des messages se répartit en quatre groupes distincts :

1. Les séquences de supervision permettent l'initialisation de la transmission dans l'un ou l'autre sens, ce qu'on appelle:

   - scrutation (Polling) si l'information va de la station vers le central;
   - sélection (Selection) si l'information va du central vers la station.

   Sous TMM-RB, seule le central a l'initiative de telles séquences.

   Les séquences de supervision sont :

   - scrutation : POL C2 ENQ ,
   - sélection : SEL C2 ENQ ,
   - alarme : SEL BEL ENQ ,

   auxquelles les réponses correspondantes sont :

   - acceptation d'une scrutation: envoi du premier bloc d'information utile,
   - refus de la scrutation : EOT ,
   - acceptation d'une sélection : SEL ACK,
   - refus de la sélection : SEL NAK,

   En cas de non-réponse de la station lors de la sélection initiale, le central émet une séquence d'alarme pour attirer l'attention de l'opérateur de la station. Elle requiert l'intervention de celui-ci pour rendre opérationnelle la station. Celle-ci doit répondre positivement ou négativement à toute séquence de supervision consécutive à la séquence d'alarme.
   Si, en phase de scrutation, la station n'a momentanément plus rien à transmettre, elle envoie la séquence EOT au central en réponse à un accusé de réception positif.
   En phase de sélection, la station ne peut faire qu'une demande de suspension positive après réception d'un bloc correct.

2. Les blocs d'information utile sont constitués d'un nombre entier d'articles encadrés par des caractères

spéciaux de procédure. Ces articles peuvent concerner des périphériques différents, toutefois on ne peut trouver qu'un seul article relatif au clavier de la console qui doit alors être placé en tête du bloc.

	SEL			ETB	
SOH		NOB	STX ... suite d'articles		BCC
	POL			ETX	

Fig. IV.3 Format des blocs d'information.

Dans la figure IV.9,
- NOB indique le rang modulo 8 du bloc dans le train des blocs émis ;
- ETX n'est employé que pour le dernier bloc du train;
- BCC est le caractère de parité longitudinal calculé sur tout le bloc, de SOH exclus à ETB, ou ETX le cas échéant, inclus.

Chaque article possède son propre en-tête de procédure et il est susceptible d'être compacté, les blancs non significatifs de fin d'article sont alors éliminés.

Le second caractère de l'en-tête, C2 ou C3, distingue les deux cas suivants :

- article non compacté :

  C2 C3 ...texte... RS

  le texte a alors un format standard, c'est-à-dire 80 caractères pour le lecteur de cartes, 133 pour l'imprimante et 72 pour la console. Le caractère RS est un séparateur d'articles;

- article compacté :

  C2 C3 C4 C5 ...texte... RS .

  Les quatres bits de droite des caractères C4 et C5 indiquent alors la longueur effective de l'article.

En transmission ASCII, aucun caractère de contrôle n'est admis dans la partie information utile de sorte que le problème de transparence n'existe pas.

3. Les accusés de réception sont émis par la station

réceptrice en réponse à un bloc d'information utile. L'accusé de réception sera positif, ACK, si aucune erreur n'a pu être détectée et négatif, NAK, dans le cas contraire. Il comporte le numéro du bloc modulo 8.

ACK
NOB
NAK

4. Les demandes de suspension se substituent aux accusés de réception lorsque pour un motif quelconque, la station ne peut plus continuer sa fonction. La demande peut être positive si le dernier bloc a été correctement reçu. Le central peut aussi émettre une demande de suspension sans indication concernant le bloc reçu précédemment : la station interprète cette demande comme étant négative.

ACK
DLE
NAK

3.3.2. Les échanges de messages

Pour ne pas alourdir la description des échanges, seule la procédure du côté de la station sera décrite, en supposant que les échanges s'effectuent dans des conditions presqu'idéales où les échanges ne subissent que des altérations détectables par la protection contre les erreurs, les problèmes de sécurité et de reprise étant explicités plus loin.

Dans ces conditions, la figure IV.10 résume la procédure de la station. Les blocs soulignés sont ceux que la station reçoit. P0 est l'état de repos, P1i sont les états de scrutation et P2j ceux de sélection.

3.3.3. Les erreurs de transmission

Un bloc d'information est invalide si les caractères d'encadrement, i.e. SOH, ETB ou ETX, sont incorrects ou n'ont pu être reconnus. La réception d'un tel bloc n'entraîne aucune réponse de la part de la station. Le central règle alors entièrement les conditions de reprise. Tout bloc est invalide s'il n'est pas conforme à la structure de la réponse attendue.

Un bloc d'information est incorrect dans les cas suivants :
- erreur de parité ,

- erreur de numérotation,
- caractère de service, autre que d'encadrement, à structure incorrecte,
- erreur de comptage de caractères dans un article.

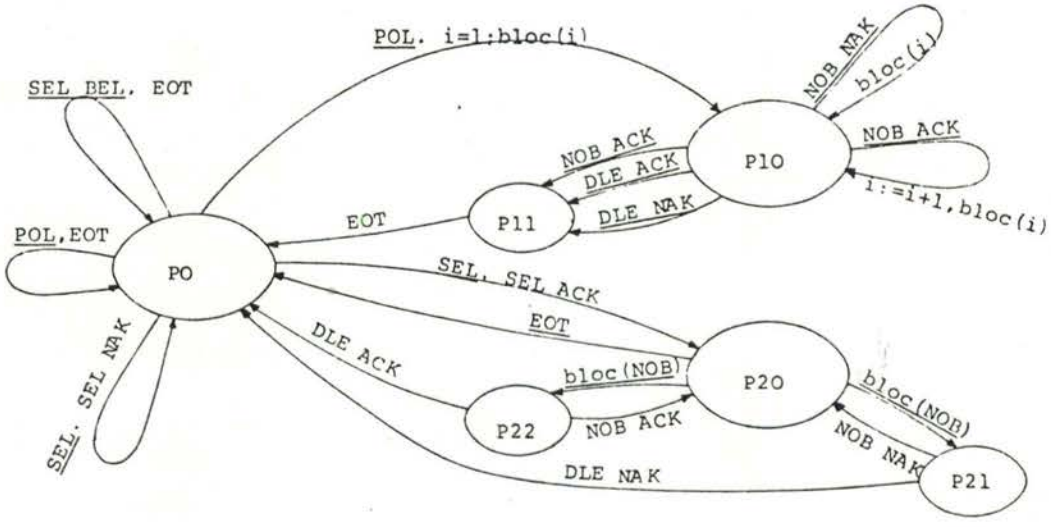


Fig. IV.4 Graphe de la procédure de la station.

Si en phase de scrutation, la station n'a momentanément plus rien à transmettre, elle envoie la séquence EOT au central en réponse à un accusé de réception positif.
En phase de sélection, la station ne peut faire qu'une demande de suspension positive après réception d'un bloc correct.

La station dispose d'une temporisation de 1,5 secondes armée à chaque fin de réception d'une séquence quelconque et désarmée au début de chaque émission. En réponse à une séquence reçue, l'émission de la station doit donc avoir lieu avant l'épuisement de cette temporisation, sinon la station doit rester silencieuse en attendant le message suivant du central.
Le central dispose en plus de cette temporisation, d'un nombre maximum p de déclenchements de cette temporisation et d'un nombre maximum m de réceptions d'un bloc invalide ou incorrect en phase de scrutation, ou de réception d'acquittements négatifs ou incorrects en phase de sélection. Le central réémet le même accusé de réception, en scrutation, ou le même bloc d'information, en sélection, tant que l'un des maxima n'est pas atteint. Quand l'un ou l'autre est atteint, le central émet :

- en scrutation, une demande de suspension négative à laquelle la station peut répondre par EOT;
- en sélection, une séquence EOT;

3.3.4. Critique de la procédure TMM-RB

Le protocole TMM-RB est indépendant de l'information utile transmise. Il assure une transmission correcte des blocs sans duplication ni répétition et dans le même ordre à la réception qu'à l'émission.

Toutefois, certaines limitations apparaissent :

- Le taux d'utilisation de la ligne est limité à 50 % de sa capacité maximum du fait du mécanisme à l'alternat qui sépare nettement la phase de scrutation de celle de sélection.

- Pour pouvoir émettre un nouveau bloc, l'émetteur doit attendre l'acquittement positif; pour pouvoir réémettre le même bloc, il doit attendre soit un acquittement négatif, soit le réveil.

- Le nombre important de caractères de service nécessaires au fonctionnement du protocole limite le rendement utile.

- Les messages de service sont moins bien protégés que les messages d'information puisqu'ils ne subissent aucun contrôle de parité.

- La diversité des structures des messages échangés complique la génération des messages à émettre et la reconnaissance des messages reçus.

Chapitre 4

4. L'APPLICATION DU LABORATOIRE

Après avoir longuement décrit les principes d'une procédure TMM-RB, nous en venons à la résolution même du protocole. Ayant déjà amplement décrit l'environnment de travail, nous ne parlerons ici que de l'architecture du logiciel devant gérer la communication.

4.1. Découpe en couches

Comme il est commun dans les problèmes de protocole, une découpe en couches (layers) du protocole a été adoptée; l'architecture retenue n'est pas non plus fortuite et s'inspire de l'habitude prise dans ce domaine où des standards [1] commencent à s'établir.

L'architecture (Fig. IV.5) est composée de quatre couches ou niveaux qui sont:

- couche application ,
- couche transport ,
- couche liaison de données ,
- couche physique .

Voyons comment ces couches interagissent entre elles et commençons dans l'ordre inverse par la couche physique.

4.1.1. La couche physique

Nous revoilà dans le hardware : il s'agit évidemment ici du câble de transmission lui-même qui relie les différents équipements entre eux ; on y retrouve également les différents interfaces de communication, de gestion des interruptions conséquentes aux évènements extérieurs. Les constituants de cette couche sont commandés par la couche directement supérieure,

---

(1) Voir aussi : [1] Voir [ANSI], [DESJARDINS] et [JACOBSEN]

à savoir la couche liaison de données, qu'ils servent.

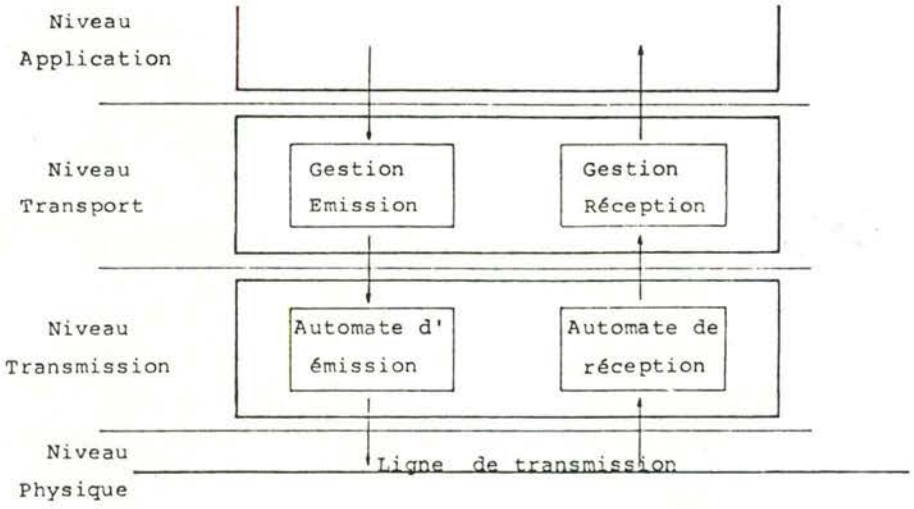


Fig. IV.5 Architecture en couches.

La fonction de cette couche est d'assurer la transmission des données individuelles entre les deux sites au moyen du câble qui les relie. La transmission s'effectuer en logique inversée, une tension de -12 V. fournissant un signal logique "1", et une tension de +12 V. un signal logique "0".

4.1.2. La couche liaison de données

C'est la couche inférieure de l'architecture du logiciel de communication.

- Elle comprend les différentes fonctions de transfert de données entre sites géographiquement distincts ;
- les données y sont généralement regroupées et encadrées par des caractères spéciaux pour former un paquet.

L'objectif de cette couche est de détecter, et si possible de corriger, les erreurs qui peuvent se produire au niveau physique.

4.1.2.1. Composition de la couche liaison de données

La couche liaison de données peut être divisée en deux automates :

- un qui assure l'émission des données, l'automate de sortie,

- un qui s'occupe de la réception de celles-ci, l'automate d'entrée.

Ces automates sont normalement inactifs et ne sont réveillés que par des évènements bien précis qui sont :

- un caractère vient d'arriver et est disponible dans l'interface pour traitement ; c'est l'évènement caractère-reçu ;
- le caractère fourni à l'interface a été transmis, et celui-ci est prêt à en recevoir un autre pour envoi; c'est l'évènement émission-prête.

Ces interruptions proviennent directement de la carte interface (ISBC 544) et sont gérés par le système d'exploitation temps réel qui les aiguille vers le serveur correspondant.

A cause du principe de l'alternat, ces deux automates ne seront jamais actifs simultanément puisque les stations sont maîtres ou exclaves tour à tour. Les interactions existant entre ces deux automates sont représentées en figure IV.6.

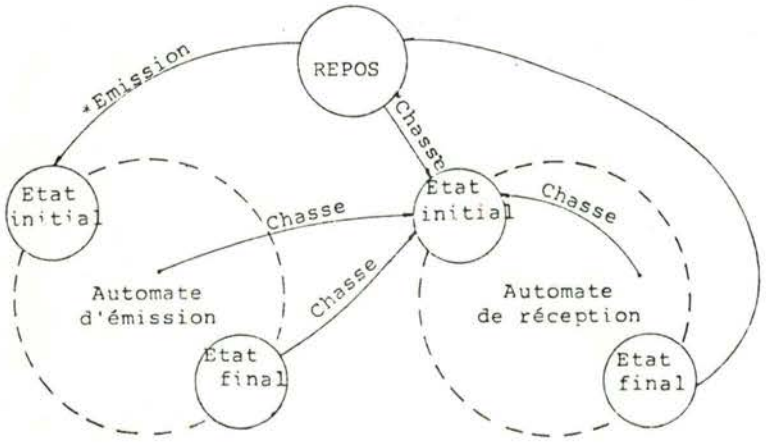


Fig. IV.6 Graphe de la procédure de transmission.

Les messages précédés d'un "*" sont des commandes de contrôles issues de la couche supérieure, le niveau transport, qui requiert des services du niveau transmission. Il en est de même pour celui-ci qui "commande" les services nécessaires à la couche physique, en l'occurence à la carte interface; on retrouve dans ces commandes, des demandes d'émission, de réception ou d'attente de réception (appelées communément mode chasse (Hunting mode)).

L'automate de la tâche de transmission (Fig. IV.7) comprend, lui, cinq états qui définissent les types de paquets en provenance du central et attendus par la station. On y retrouve donc :

1. L'état de repos, où la station attend une demande soit de polling, soit de sélection depuis le central ;

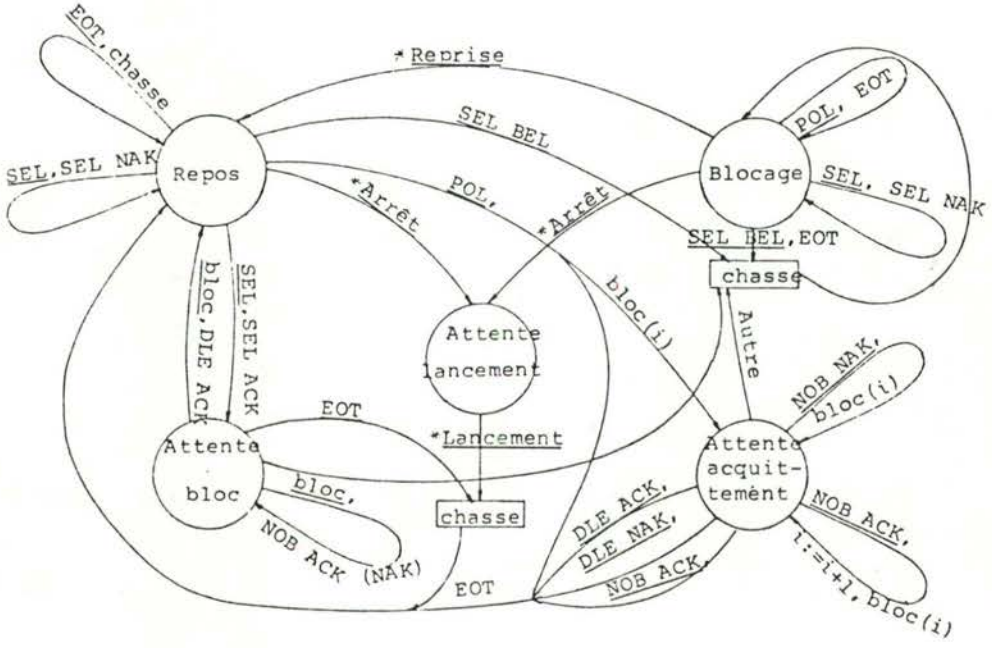


Fig. IV.7 Automate de la tâche de transmission.

2. L'état d'attente de blocs, où, en phase de sélection, la station attend de recevoir un bloc de données en provenance du central ;

3. L'état d'attente d'acquittement, où, en phase de polling, la station attend l'acquittement par le central d'un bloc émis par la station;

4. L'état de blocage, où, suite à la commande d'alarme du central (SEL BEL), on refuse toute commande de scrutation ou de sélection avant d'avoir été "débloqué";

5. L'état d'attente de lancement, où la station est en fait déconnectée de la ligne jusqu'à l'autorisation d'initialisation de celle-ci.

On remarque que cet automate est sous contrôle intégral de la couche supérieure via les commandes (ordres précédés d'un "*") :

- Lancement : qui initialise la ligne en attente d'une demande de polling ou de sélection en provenance du central,

- Arrêt : qui permet de couper la liaison logique établie entre les deux sites; le rétablissement de la liaison se fera par un nouveau "Lancement", et

- Reprise : qui repositionne l'automate dans son état de repos.

D'autre part, l'automate requiert les services de la couche inférieure via des demandes :

- d'envoi de blocs d'information ou d'acquittement ;
- de réception de blocs d'information ou d'acquittement ;
- d'attente de réception de blocs ou d'acquittements.

4.1.2.2. L'automate d'entrée

L'automate d'entrée , ainsi d'ailleurs que l'automate de sortie, fonctionne sous interruptions. Il faut se souvenir qu'on travaille dans un environnement en temps réel qui permet d'exécuter certaines tâches du système pour servir les différents évènements qui peuvent se présenter.

Dans ce cas précis, l'automate d'entrée commande à l'interface de recevoir un caractère, et désire être averti, i.e. être réveillé, lors de la disponibilité de celui-ci. L'arrivée du caractère provoquera l'évènement caractère-reçu en provenance de l'interface. Le gérant des tâches, averti de cet évènement, passe la main au service correspondant, qui, en l'occurence, est la procédure d'entrée.

Au fur et à mesure que l'automate auquel celle-ci est associée reçoit les caractères d'un message en provenance du central, il passe d'un état à un autre , en ayant commencé de l'état initial, pour aboutir finalement à l'état final.

La station ne devant répondre que si le paquet reçu est reconnu (un message de données erroné est reconnu, mais pas un message de service dont la structure est incorrecte), tout caractère non admis par l'automate dans l'état où il se trouve lorsqu'il reçoit celui-ci, provoque son retour à l'état initial et par là-même, le retour en mode chasse de l'interface.
De plus, suivant l'état de l'automate, attente de bloc ou d'acquittement, la tâche reconnaîtra ou non les messages de type bloc ou acquittement, outre ceux de commande (Polling, sélection, suspension, fin de transmission,...)

La procédure d'entrée

Lorqu'un caractère est disponible c'est-a-dire, lorsque l'interruption associée à l'évènement caractère-reçu est apparue, la procédure d'entrée (Fig. IV.8) associée à l'automate est activée; celle-ci effectue un prétraitement qui est selon le cas:

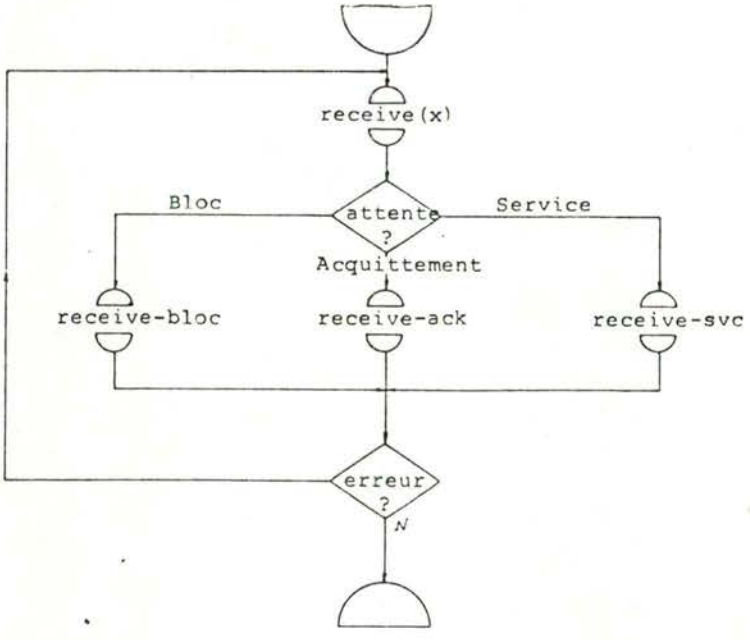


Fig. IV.8 Procédure d'entrée.

- Remise de l'automate à l'état initial lorsque le caractère reçu correspond au code PAD (ASCII); ceci est dû à la particularité de l'interface qui reconnait des caractères bien que la ligne de transmission soit inactive (aucun caractère ne circule). Il reconnait donc une suite de huit bits à "1" soient :

  1 1 1 1 1 1 1 1 (FF hex).

  L'interface effectue sur chaque caractère reçu un contrôle de parité, et fait la correction si nécessaire. Dans le cadre de cette application, on utilise une parité dite impaire, le nombre total de bits à "1" étant impair. Dans le cas du caractère PAD , l'interface signalera une erreur de parité, et fournira le caractère codé sur 7 bits, soit :

  0 1 1 1 1 1 1 1 (7F hex).

  On reconnaîtra un caractère PAD par :

a. sa valeur 7F en hexadécimal et par
b. l'erreur de parité qu'il provoque au sein de l'interface.

- Pour les autres cas d'erreurs de parité ou d' overrun (bourrage du buffer de réception de l'interface), on remplace le caractère reçu par un caractère à priori faux qui sera transmis à l'automate comme caractère effectivement reçu, mais qui provoquera, du moins on l'espère, une erreur sur le contrôle de parité longitudinale. Le message reçu ne sera donc rejeté qu'après réception complète, soit lorsqu'on aura rencontré le premier des deux caractères ETB ou ETX.

- Les caractères de remplissage SYN , qui, servent à la synchronisation des deux sites impliqués dans le dialogue, ne doivent pas être traités.

Lorsque le pré-traitement est fructueux, le caractère reçu est transmis à l'automate qui, après l'avoir sauvé dans une zone prévue à cet effet et dont l'adresse a été spécifiée par le niveau transport, en effectue le traitement proprement-dit. Celui-ci consiste :

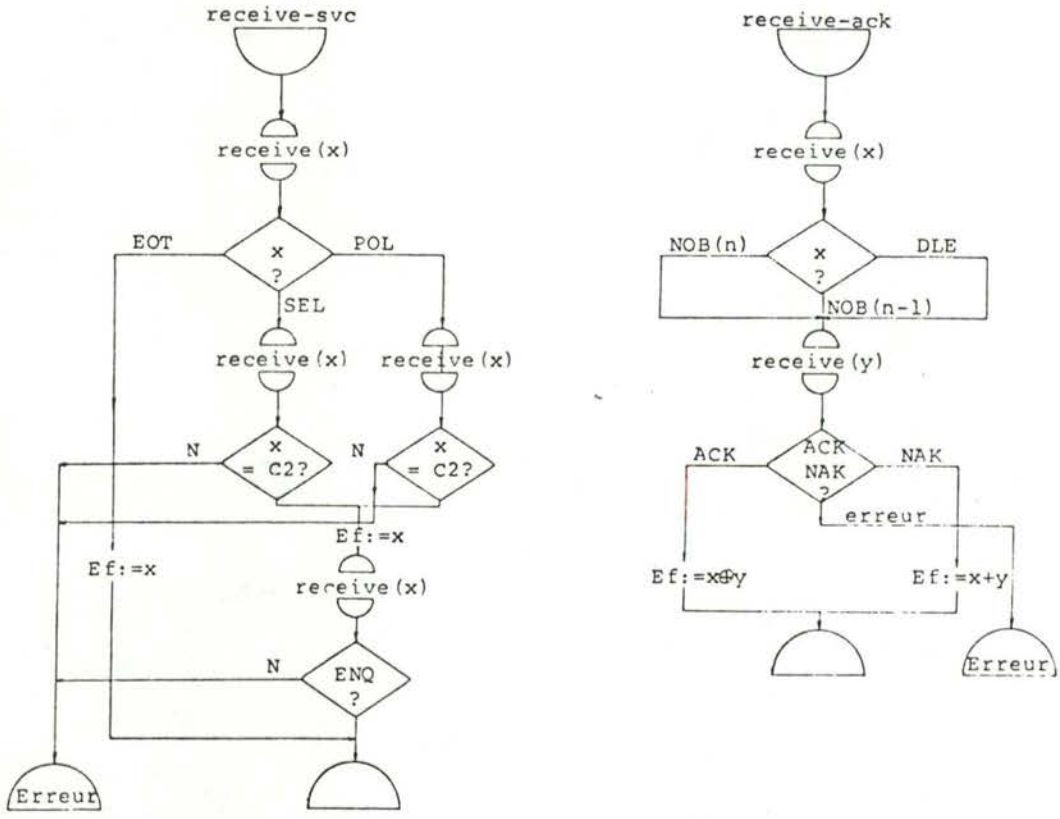


Fig. IV.8a. et IV.8b.

- En réception de bloc (Fig. IV.8a), dans la vérification de sa longueur qui ne peut en tous les cas pas excéder 256 bytes, dans le calcul de la parité longitudinale du bloc et dans l'analyse de son format qui doit être du type :

SOH SEL NOB(n) STX [C2 [C3] ...article [RS ...] ...] ETB BCC

- En réception de message d'acquittement (Fig. IV.8b), en la reconnaissance de son format qui doit être un des suivants :

NOB(n)	ACK
NOB(n-1)	NAK
DLE	ACK
DLE	NAK

- En réception de message de service (Fig. IV.8c), dans la reconnaissance d'un des formats qui suivent :

EOT		
SEL	C2(SEL)	ENQ
POL	C2(POL)	ENQ

Lorsqu'un message, de données, d'acquittement ou de service, a été entièrement reçu, la couche liaison de données avertit la couche transport de l'exécution de son service et des conditions de réalisation de celui-ci. Celles-ci sont transmises sous la forme d'un paramètre dont la valeur est celle de l'état final de l'automate, qui, selon le cas, est :

- EOT : arrêt de communication (04H) ,
- C2(SEL) : sélection du périphérique C2 (2FH, 60H, 61H, 62H, 63H) ,
- C2(POL) : polling du périphérique C2 (40H, 41H, 42H) ,
- NOB/ACK, NOB/NAK, DLE/ACK, DLE/NAK : (respectivements représentés par les codes 8nH, CnH, 88H, C8H, où n est le numéro du bloc envoyé) s'il s'agit d'un acquittement de bloc,
- ETB ou ETX : s'il s'agit d'une réception de bloc correct (17H, 03H),
- ? : s'il s'agit d'une réception de bloc incorrect (le code correspondant est 15H).

Enfin, la procédure restaure son état initial pour une prochaine activation.

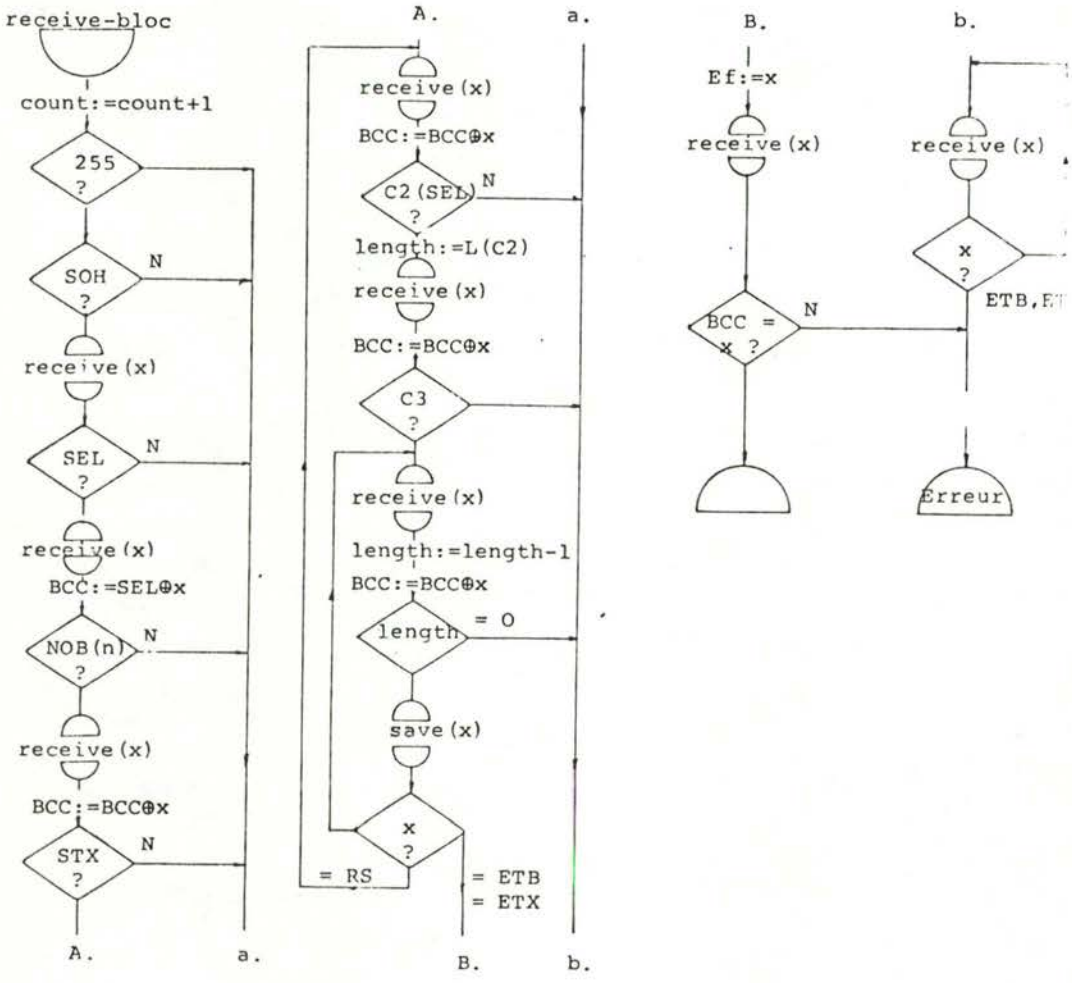


Fig. IV.8 c.

Variables :

Ef : Etat final de l'automate.
BCC : "Block Control Character".
count : compteur.
length : longueur de l'article.
x,y : caractère reçu.

Procédures :

receive(x) : recevoir un caractère; x contient le caractère reçu.
save(x) : sauver le caractère reçu dans le buffer.

Fig. IV.8 c.

4.1.2.3. L'automate de sortie

Cet automate s'exécute sous interruptions. Il reçoit le message à émettre de la couche transport, l'encadre des caractères de contrôle qui dépendent de son type (données, acquittement ou service), et il transmet un à un les bytes du message à l'interface, au niveau physique, jusqu'au dernier, y compris le byte de parité longitudinale, et termine de la sorte son service.

Quant au niveau physique, il signale, par interruption, que son service est terminé, i.e. que le caractère a été envoyé et que lui-même est prêt à en recevoir un autre; à la suite de cet évènement, émission-prête, l'automate lui transmet le byte suivant à émettre.

La procédure de sortie

Cette procédure (Fig. IV.9) réalisera donc la fonction de l'automate auquel elle est associée.
Celle-ci ne connait en fait que deux types de messages : les blocs de service, où on retrouve aussi bien les accusés de réception que les séquences de supervision, et les blocs d'information.

- L'ensemble des messages de service est donc :

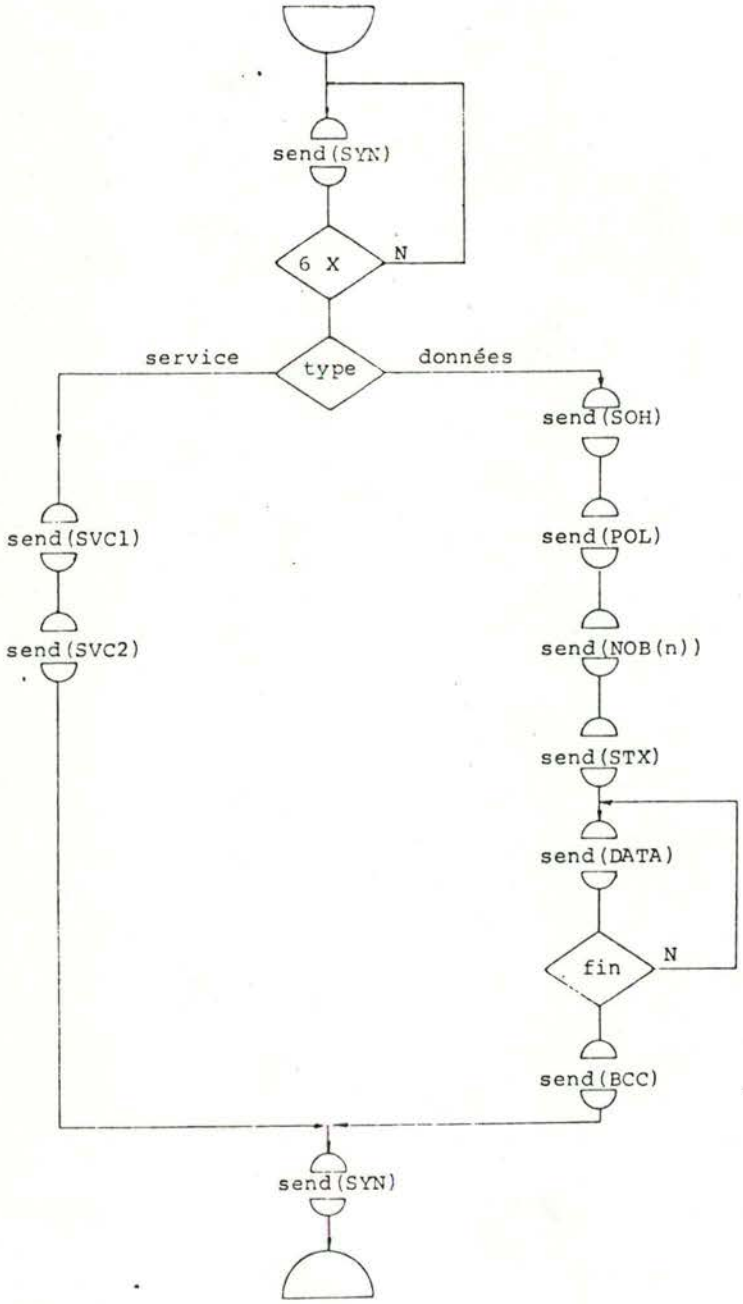
EOT	
SEL	ACK
SEL	NAK
DLE	ACK
NOB(n-1)	ACK
NOB(n-1)	NAK

- Un message d'information aura toujours le format suivant :

SOH POL NOB(n) STX ..texte.. ETB(ETX) BCC

- Dans les deux cas, le bloc de données qui comprend le message à envoyer, de quelque nature qu'il soit, est une zone mémoire dont l'adresse est fournie à la procédure lors de sa première invocation.

Le travail exigé par la procédure de sortie est beaucoup plus simple que pour la procédure d'entrée. Il consistera dans :



Procédure :

send(x) : envoyer le caractère x.

Fig. IV.9 Procédure de sortie.

- l'encadrement des messages à envoyer par les caractères de contrôle adéquats ;

- le calcul de la parité longitudinale des messages d'information qui sera transmise avec le dernier byte.

- l'envoi de tous les bytes du messages donnés (un à deux bytes pour les messages de service et jusqu'à 252 bytes pour ceux d'information);

A la fin d'une émission, la procédure restaure son état initial pour une activation ultérieure.
A cause du principe de l'alternat (les procédures d'entrée et de sortie ne sont jamais actives simultanément), la procédure de sortie ne peut pas garantir la bonne réception du message envoyé avant d'en avoir reçu l'accusé de réception. Ce dernier ne peut être réceptionné que par la procédure d'entrée. C'est pourquoi, lorsque l'automate de sortie a fini sa transmission, il passe la main à l'automate d'entrée; on peut d'ailleurs voir ceci sur la figure IV.7. C'est donc le résultat de la réception consécutive à une émission qui servira comme résultat de la bonne exécution de celle-ci à la couche supérieure.

4.1.3. La couche transport

Souvent appelé transport de bout en bout, cette couche doit vérifier l'intégrité de la couche liaison de données, aucun système de transmission n'étant totalement fiable. Cette couche comprend donc le protocole nécessaire pour assurer un transfert correct, i.e. sans erreurs, des données par la couche liaison de données, et s'occupe, le cas échéant, des corrections ou des retransmissions.

Les fonctions réalisées par la couche transport sont:

- la détection des erreurs et la gestion de la qualité du service;
- l'éclatement des blocs en articles;
- les fonctions de supervisions du service.

4.1.3.1. Communication inter-processus

La couche transport s'appuye donc sur les services, émission et réception, fournis par la couche liaison de données : la tâche qui exécute les fonctions du protocole de transport peut en effet requérir les services des procédures

de transmission, en demandant au gérant des tâches de susciter l'arrivée d'un des évènements extérieurs caractère-reçu ou émission-prête. Les deux procédures de transmission étant automatiquement repositionnées à leur état initial à chaque fin de travail, le service requis pourra donc reprendre depuis son début.

De même, lorsque le service transmission se termine, et on l'a vu, toujours après une réception, la procédure d'entrée avertit le niveau transport de la fin de son travail; Ceci est réalisé par le biais d'évènements associés à une file d'attente.

L'évènement, en fait un message interne, est fourni au gérant des tâches ainsi que la file d'attente (appelée canal dans le système temps réel RMX) à laquelle il est associé; c'est au gérant d'assurer la transmission du message à la première tâche qui le désire, en la débloquant si nécessaire.
Seules les tâches actives ou celles sous interruptions, peuvent donc provoquer pareils évènements, l'arrivée de chacun de ceux-ci pouvant causer alors soit le déblocage d'une tâche en attente, plus prioritaire que la tâche qui a provoqué l'évènement, soit la continuation de la tâche courante, si elle est elle-même la plus prioritaire des tâches prêtes.

Deux procédures permettent la communication entre tâches. Ce sont:

- RQSEND (canal_addr,msg_addr) : envoyer le message dont l'adresse de début de zone mémoire est msg_adr sur la file d'attente dont l'adresse est donnée par canal_addr.

- x=RQACPT (canal_addr) : attendre l'arrivée d'un message sur le canal spécifié; comme il s'agit ici d'une fonction, le résultat x sera l'adresse en mémoire de la zone qui contient le message reçu.
  Si un message est déjà disponible, il n'y aura pas d'attente; si aucun message n'est disponible, la tâche est désactivée et bloquée.

Il y a en outre une troisième procédure, fort semblable à RQACPT, mais qui permet à une tâche, sans être bloquée, de savoir si un message est disponible sur un canal :

- RQWAIT (canal_addr,time_limit) : attendre pendant time_limit * 25ms l'arrivée d'un message sur le canal dont l'adresse est spécifiée. Si time_limit vaut 0, on obtient le cas particulier sans attente.

4.1.3.2. L'interface transport-transmission

L'interface entre les deux couches transport et transmission se base sur ce mécanisme de communication inter-processus :

La procédure de transport informe , via le gérant, qu'elle a besoin des services d'une des deux procédures de transmission. Pour celà, elle lui transmet l'adresse de la zone mémoire qui soit contient le message à transmettre, soit qui est destinée à recevoir le message reçu. Ensuite, elle attend la signification de la bonne exécution de sa demande, et, pour celà, fait une demande de réception de message interne sur un canal (Kintsyn) réservé à cet effet et commun aux deux couches.

4.1.3.3. L'interface application-transport

L'interface entre les deux couches application et transport, est composé de 7 canaux et de 2 variables booléennes.

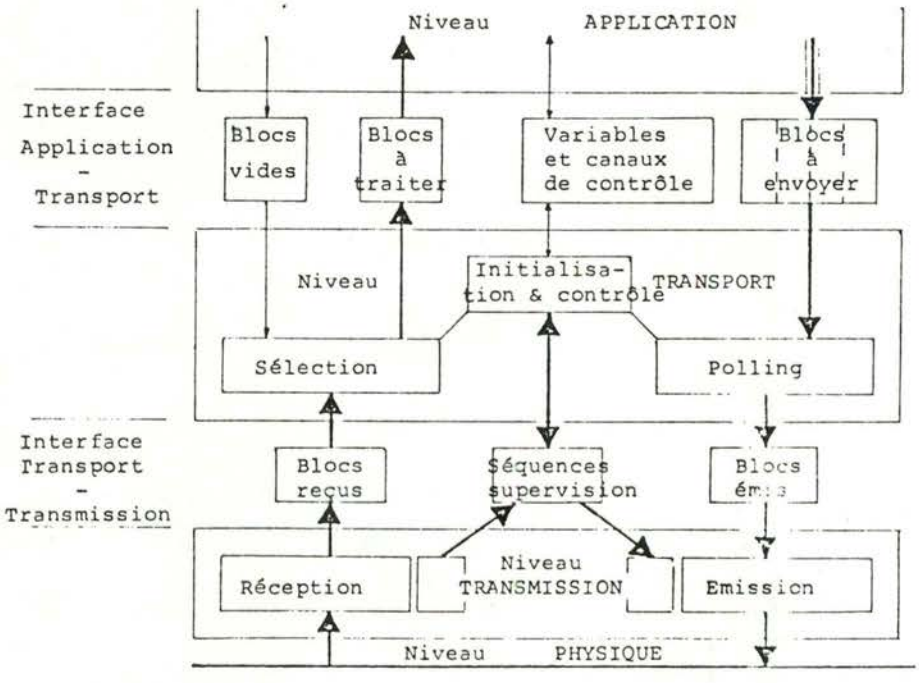


Fig. IV.10 Interfaces inter-couches.

Les canaux

Le gérant de la ligne a besoin de

- zones mémoires où il peut mettre les messages en cours de réception;
- zones mémoires où il peut prendre des messages à envoyer;
- zones mémoires où il peut renvoyer les messages complètement reçus aux tâches d'application qui se chargeront de leur traitement.

Ces zones mémoires sont des buffers de longueur suffisante pour contenir un message quelconque servant à la liaison. On obtient ainsi les canaux suivants :

1. Canal de réservation de la ressource synchrone (Kresres) sur lequel le programme d'application fait sa demande de ressource. Lorsqu'il l'a reçue, il doit renvoyer le message sur le canal des interruptions (Kintsyn) en ayant pris soin d'en modifier le type de telle sorte que la tâche de transport reconnaisse là une demande d'initialisation de la liaison; le type de ce message sera typ$trig$syn.

2. Canal d'interruption du gérant (Kintsyn), sur lequel un programme d'application peut envoyer :

   - le message d'initialisation de la liaison après réservation de la ressource;

   - un message de déblocage après erreur : ce message peut être soit une demande de libération de la ressource, auquel cas il sera du type typ$free$syn, soit une demande de reprise de communication normale, et il aura alors le type typressyn;

   - le message de demande d'arrêt des communications (typ$stop$syn) qui permet de libérer la ressource à tout moment et non plus sur erreur. Une réponse à ce message sera envoyée sur un canal spécifique avec un type dépendant de l'action :

     - erreur (typerrblocked) : indiquant qu'il faut d'abord débloquer la ligne,
     - arrêt (typ$stop$syn) : indiquant que la ressource a été libérée.

3. Canal d'erreurs (Kerrsyn) : c'est sur ce canal que le gérant de la liaison renvoie les messages d'erreur. Deux sortes de messages peuvent y être envoyé :

   - message d'anomalie (typ$wrong$syn) en cas d'anomalies des messages reçus du central, et
   - message d'alarme (typ$alarm$syn) en cas de réception d'une séquence d'alarme (SEL BEL) du central.

Suite à l'un de ces messages, le gérant de la ligne attend de recevoir sur le canal de contrôle (Kintsyn) un message d'arrêt ou de reprise (Voir 2.). Tant qu'un de ces deux messages n'est pas reçu, la ligne reste bloquée et le gérant répond négativement à toute demande du central.

4. Canal des blocs libres (K$free$rec) : la tâche y prend les buffers pour y copier les messages entrant; si aucun buffer n'est disponible, c'est-à-dire que le canal ou la file d'attente est vide, elle ne peut copier le message qui arrive et renvoie pour celà un accusé de réception négatif (SEL NAK) à la demande de sélection du central.

5. Canal des blocs à traiter (K$treat$rec) où les différents programmes d'application prennent des blocs en vue de leur traitement. Les blocs qu'ils pourront y trouver seront de plusieurs types suivant la sélection éffectuée :

   a. Type console (typreckeyboard) : ceux qui ne contiennent que des articles de 72 caractères au plus;
   b. Type imprimante (typrecprint$ETB ou typ$rec$print$ETX) : ceux qui contiennent des articles console (72 c.) ou imprimante de 132 caractères au plus. Il y en a deux types, suivant qu'il s'agit ou non d'une fin de texte (ETB ou ETX). Si c'est le cas, l'imprimante n'est plus validée.
   c. Type perforateur de cartes (typreccard$ETB ou typ$rec$card$ETX) : ceux qui contiennent des articles soit de console (72 c.) soit à destination du perforateur de cartes avec une longueur d'au plus 80 caractères. Ici aussi on fait la différence avec la fin de texte : dans les mêmes conditions que ci-dessus, le perforateur de carte sera invalidé s'il s'agit d'une fin de texte (ETX).

Canal des blocs à envoyer : lorsque le central fait une demande de scrutation (Polling), la station vérifie qu'elle a au moins un bloc de données à lui envoyer. Pour celà, elle regarde si la file d'attente des blocs à envoyer est vide ou non. Cette file d'attente est en fait dédoublée.

Ceci est nécessaire par le fait que plusieurs périphériques peuvent vouloir émettre des blocs d'information; il y a un canal réservé pour chacun des périphériques suivants:

- la console utilisateur,
- le lecteur de cartes,

6. Canal des blocs urgents (K$prior$send) où la tâche qui gère la ligne prend les blocs à émettre en priorité; ce sont les blocs en provenance de la console uniquement.

7. Canal des blocs normaux (K$norm$send) où la tâche gérant la ligne prend les blocs à émettre soit lorsqu'il n'y en a pas sur le canal urgent, soit que le central en ait fait la sélection explicite. Ce canal est en effet uniquement réservé aux blocs du lecteur de cartes; on peut cependant y trouver éventuellement des blocs en provenance de la console.

   On remarque donc la différence avec le canal des blocs à traiter où tous les blocs sont mélangés, ne tenant pas compte du périphérique auquel ils sont destinés.

Les messages

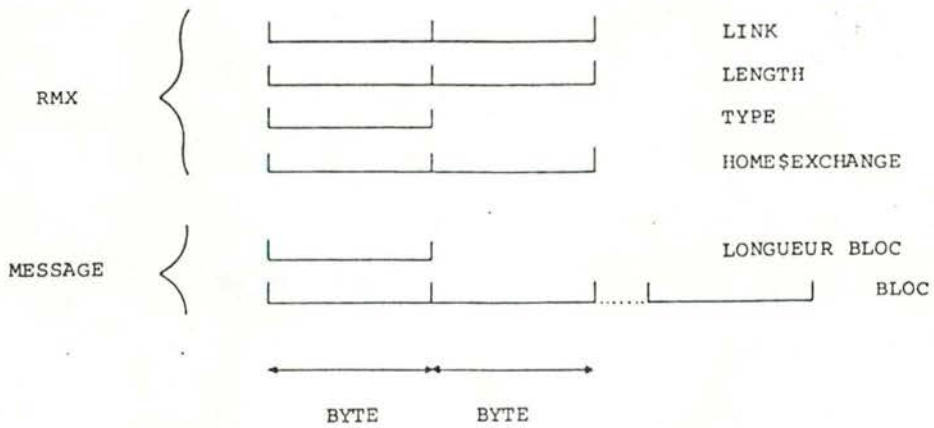


Fig. IV.11 Structure des messages.

Les messages qui circulent sur ces canaux ont une structure bien définie et ceux qui sont déposés sur le canal des blocs libres sont supposés avoir une longueur suffisante pour contenir un bloc de 252 octets.
Leur structure est donnée en figure IV.11 : chaque message est composé d'une partie réservée au noyeau du système (RMX) et d'une autre partie utilisée pour la communication.

Seul le message de réservation de la ressource synchrone a une structure différente où sont spécifiées les adresses des canaux utilisés pour la communication (Fig. IV.12).

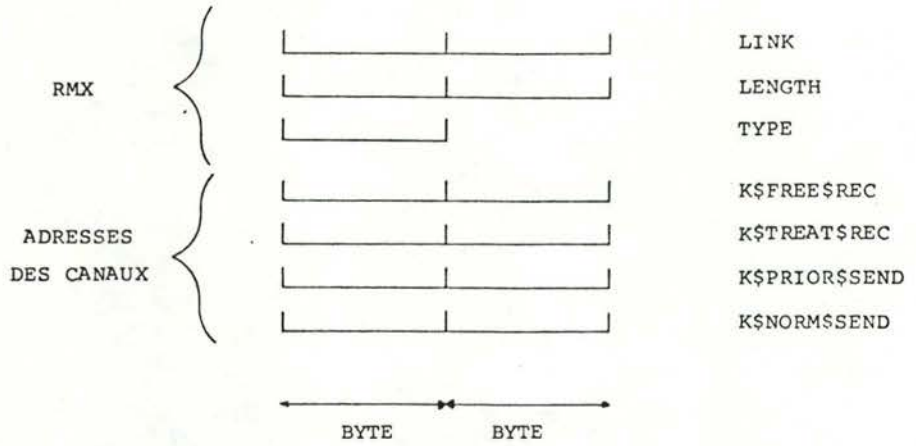


Fig. IV.12 Structure du message de réservation de la ressource.

Les variables de contrôle

Les variables de contrôle, qui autorisent l'accès à l'imprimante ou au lecteur-perforateur de carte, sont des variables booléennes (printer$OK et puncher$OK) dont la valeur vraie (True) spécifie que le périphérique correspondant est validé. Ces variables sont automatiquement repositionnées à la valeur faux (False) dès réception de la fin de texte (Bloc terminé par ETX).

La figure IV.10 montre la constitution des interfaces inter-couches : on notera que les buffers de réception et d'émission ne sont pas des buffers supplémentaires. Suivant la seconde considération du point 2.2, il fallu réduire au maximum non seulement la taille du code, mais aussi celle des zones de travail qui sont implantées, elles, en mémoire vive. C'est pourquoi les automates de transmission n'ont pas de zones de travail propres mais emploient directement les buffers libres et ceux à envoyer, en recevant de la couche transport les adresses respectives de ces buffers. Ceci permet d'ailleurs de réaliser la première considération de ce même point : le logiciel de communication sera plus rapide dans son exécution puisqu'on évite ici la surcharge (overhead) qui aurait été subie si on avait dû effectuer le recopiage des blocs à envoyer dans le buffer de l'automate d'émission ou le recopiage du buffer de réception dans les blocs à traiter.

4.1.3.4. La fonction de transport réalisée sous forme de trois modules

Note préliminaire : Nous conviendrons d'adopter les terminologies suivantes :

Procédure : l'ensemble des règles qui réalisent le protocole; il s'agit donc du terme procédure au sens BSC ou HDLC et non pas au sens ALGOL.

Ressource synchrone : l'ensemble des éléments qui permettent d'établir une liaison entre les deux sites.

La fonction de transport, dont le but est d' assurer la gestion du protocole, peut être divisée en trois modules distincts :

- Module d'initialisation et d'arrêt dont la fonction est d'assurer l'initialisation de la procédure, les aspects spécifiques aux séquences de supervision, les arrêts, blocages et reprises conséquentes de la procédure;

- Module de gestion de la procédure en sélection, qui se charge uniquement de la résolution des transmissions où les informations en provenance du central sont reçues par la station;

- Module de gestion de la procédure en scrutation, qui se charge uniquement, lui, de la résolution des transmissions où les informations circulent depuis la station vers le central.

La figure IV.13 donne une représentation de ces trois modules.

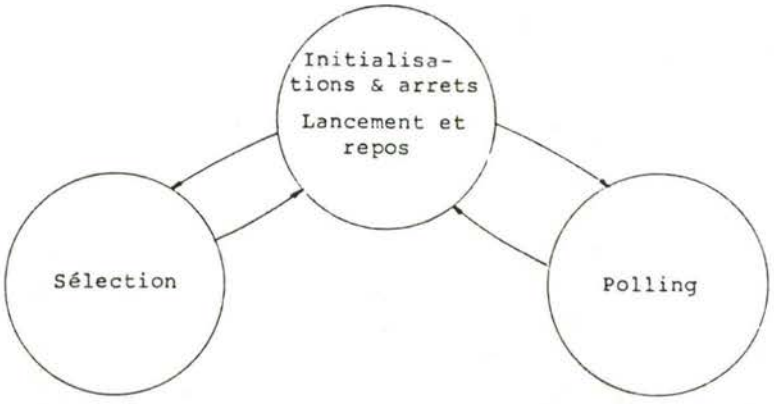


Fig. IV.13 Graphe de la procédure de transport.

4.1.3.5. Les trois modules de la procédure de transport

Nous allons maintenant parcourir chacun des modules de la couche transport :

Le module de contrôle

Ce module, dont l'algorithme est donnée en fig IV6, assure les fonctions suivantes :

1. Initialisation de la procédure (Etat initial);
2. Attente de l'ordre de lancement (Etat lancement) depuis une tâche d'application. Lorsque l'ordre est reçu, initialisation de la liaison et mise en mode chasse de l'interface (Etat repos);
3. Suivant qu'il s'agit de demande de polling ou de sélection, le traitement est dirigé vers le module correspondant.
4. Le module de contrôle peut aussi recevoir des ordres de blocage ou d'arrêt.

Le module de gestion de sélection

Dans le transfert d'information du central vers la station, ce module se charge de :

1. déterminer le périphérique sélectionné; si la sélection est faite sur un des périphériques imprimante ou lecteur-perforateur de cartes il vérifie que ce périphérique est validé, sinon il refuse la sélection (SEL NAK);
2. prendre un bloc libre dans la file d'attente : s'il n'y en a plus, il refuse la sélection de la même manière qu'en 1. (SEL NAK);
3. enfin, il peut accepter la sélection (SEL ACK) et
4. attend de recevoir un bloc d'information :
   - si le bloc est incorrectement reçu, il renvoie la séquence NOB NAK;
   - si le bloc est correctement reçu, il le met sur la file d'attente des blocs à traiter et suivant que le bloc se termine par ETB ou ETX , il renvoie la séquence de supervision NOB ACK ou DLE ACK. Il retourne à l'état de repos;
   - on a reçu une séquence de supervision EOT : c'est la fin de la sélection et il retourne à l'état de repos;
   - tout autre bloc ou séquence qui ne satisfait pas ces conditions provoque l'arrêt de la liaison.
5. retour au point 4. .

Le module de gestion de scrutation

Dans le transfert d'information de la station vers le central, ce module se charge de :

1. vérifier si on est en test de ligne (Périphérique fictif validé) auquel cas il prépare un bloc fictif pour envoi;
2. vérifier si un bloc en provenance de la console utilisateur est en attente : si oui, il le prépare, sinon il fait de même pour le lecteur de cartes;
3. si la station n'a aucun bloc d'information à transmettre, ou si le périphérique fictif n'est pas validé, il refuse la scrutation en renvoyant la séquence de supervision EOT et retourne à l'état de repos;
4. il envoie le bloc préparé et en attend l'accusé de réception;
5. si l'acquittement est négatif (NOB NAK), il recommence l'envoi du bloc (retour au point 4.), sinon il recommence au point 2. après avoir signifié à la tâche à laquelle appartenait le bloc envoyé que le service a été correctement accompli;
6. si l'acquittement d'un bloc contient une demande de suspension (DLE ACK ou DLE NAK), il retourne à l'état de repos.

4.1.4. La couche application

L'ensemble des tâches régissant la procédure de communication TMM , peut être vu comme une ressource du système que les différents programmes d'application partagent. C'est pourquoi, dans la suite du texte on parlera de ressource synchrone.
On décrira ici la façon dont un programme d'application doit utiliser la ressource synchrone.

Celle-ci est vue sous la forme de sept files d'attente, ou canaux, et de deux variables de contrôle.

4.1.4.1. Initialisation de la ligne

Bien que les tâches de gestion du protocole TMM soient toutes chargées lors du bootstrap du système, elles ne s'occupent pas tout de suite de la création et de la gestion de la liaison avec l'autre site mais se mettent d'abord en état d'attente de réception d'un ordre.

Cet ordre est donné par une tâche du niveau application à la suite duquel, la tâche transport initialisera effectivement la liaison.

En fait ceci se passe en deux étapes distinctes :

- Lors de sa création ,la tâche de transport signale sa présence au sein du système en envoyant sur un canal, réservé à cet effet et appelé canal de la ressource synchrone, un message; ensuite, elle attend l'ordre d'initialisation sur un autre canal.

- Une tâche d'application désireuse de communiquer avec l'autre site doit s'assurer que la ressource est disponible: elle attend le message de disponibilité de la ressource synchrone sur le canal correspondant; ceci fait, elle renvoie ce même message sur un canal de contrôle. La tâche de transport reconnaît celui-ci comme un ordre d'initialisation.

La ligne est initialisée à l'usage exclusif de la première tâche d'application qui a pris le message de réservation de la ressource synchrone. Il n'y a donc pas moyen d'initialiser deux fois la liaison sans que celle-ci n'eut été précédemment suspendue par demande explicite de la station, ou par détection d'erreurs non récupérables.

4.1.4.2. Utilisation de la ressource synchrone

Dès que la ressource synchrone est initialisée, tout programme d'application peut l'utiliser, soit pour envoyer des messages au central, et ce en déposant sur les canaux

d'envoi (K$prior$send ou K$norm$send) les messages voulus, soit pour recevoir des messages du central, en prenant ceux-ci sur le canal des messages à traiter (K$treat$rec).

4.2. La réalisation

Elle consiste principalement dans la conception et l'écriture des différentes procédures qui forment le protocole.

4.2.1. Les programmes

Ces programmes [Annexe B2] ont été réalisés suivant la même découpe en couches que celle abordée précédemment. C'est ainsi qu'on retrouve :

- au niveau transmission : les deux automates, celui d'émission et celui de réception, qui ont été écrits en Assembleur 8080 ;
- au niveau transport : la tâche de gestion du transport qui, écrite en PL/M 80, regroupe les trois modules directeur, de sélection et de scrutation.
- au niveau application : les programmes qui ont permis les tests de cette procédure pendant le développement du système de communication.

4.2.2. Les tests

4.2.2.1. Principe de test

Pour effectuer les tests de la ligne, il est apparu intéressant de conserver en mémoire tout le dialogue entrepris entre les deux sites pendant une session. Ceci permet de tester tant les procédures de transport (Réponses adéquates), que les automates de transmission (Encadrement des blocs, fonctionnement sous interruptions).
Un exemple de dialogue est donné en annexe (Annexe B3). Tous les caractères de contrôle d'une liaison y sont traduits par un mnémonique pour rendre le travail de vérification moins ardu. On y remarque la quantité de séquences de supervision nécessaires pour le transfert, dans un sens comme dans l'autre, d'informations utiles.

4.2.2.2. La configuration de test

Plusieurs tâches d'application ont été créées en vue des tests. Ce sont notamment :

- USER : tâche simulant la console utilisateur. Elle demande d'introduire via le clavier des commandes qui seront transmises, i.e. envoyées, au central.

- SCREEN : tâche simulant un programme d'application qui traite les blocs reçus. Elle doit, comme travail, prendre les blocs prêts au traitement, et dans ce cas-ci, afficher leur contenu utile à l'écran.

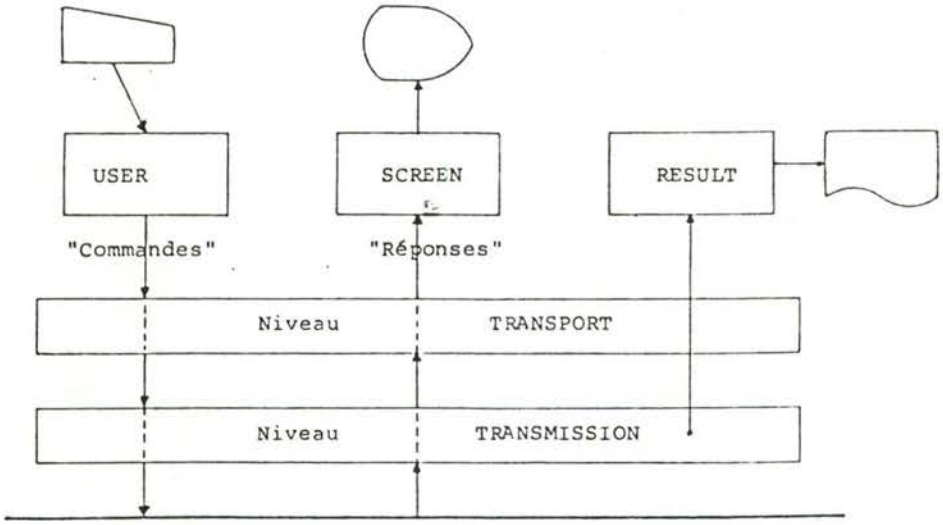


Fig. IV.14 Configuration de test.

Ces deux tâches ont leur intérêt puisqu'elles obligent les blocs qu'elles envoient ou qu'elles reçoivent, à traverser toutes les couches de la procédure.
Une dernière tâche d'application doit être rajoutée à cette configuration :

- RESULT : c'est elle qui permet de "visualiser" le dialogue qui circule sur la ligne. Elle constitue plus du "rafistolage" de test, puisqu'elle se repique directement sur le niveau transmission, sans passer par le niveau transport.

CINQUIEME PARTIE

REALISATION D'UN RESEAU LOCAL

INTRODUCTION

Pour différentes raisons, qui ont déjà été exposées dans ce mémoire (coût, disponibilité, sécurité et efficacité d'un système d'informatique répartie) il est apparu intéressant d'interconnecter entre eux, sous forme d'un réseau local, plusieurs micro-ordinateurs dont l'Institut d'Informatique dispose déjà.
Le réseau qui serait développé doit fournir des facilités de communication et d'échange de données entre ces micro-ordinateurs sur un espace géographiquement limité.

L'étude de l'implémentation d'un réseau local intervient donc directement dans le cadre de ce mémoire, la réalisation qui en dépend étant influencée par

- les besoins immédiats d'un réseau local,
- les facilités de réalisation matérielle du réseau,
- la disponibilité des éléments nécessaires à sa réalisation.

Après avoir présenté le matériel disponible et envisager plusieurs solutions pour un réseau local, on exposera la solution retenue, sous son aspect physique (Hardware) et logique (Software de communication).

Chapitre 1

1. PRESENTATION DU MATERIEL DISPONIBLE

Le matériel dont on dispose est constitué de micro-ordinateurs North-star Horizon.

1.1. Présentation du North-star

Les North-star sont des micro-ordinateurs pilotés par un microprocesseur Z80 A de ZILOG ; ils ont une capacité mémoire (RAM) de 64 Kbytes et disposent d'une mémoire de stockage (Disquettes) de 179Kbytes par unité (1 à 4 unités). Ils possèdent entre autre deux interfaces série RS 232 et un interface parallèle.

1.2. Présentation des interfaces RS 232

Chaque North-star comprend deux interfaces RS 232 (Avis V.24 du CCITT) dont un est réservé à la connexion du terminal, le second étant dès lors disponible pour d'autres applications, par exemple un réseau local. Chaque porte RS 232 fournit plusieurs signaux dont notamment :

- un signal de données en entrée (Received Data) ;
- un signal de données en sortie (Transmitted Data) ;
- un signal de référence ou masse (Ground signal) ;
- un signal d'état de modem (Data Set Ready).

Ces interfaces peuvent travailler en full duplex (Voir infra pour l'utilité de ceci), en mode synchrone ou asynchrone.

Les standards s'y rapportant limitent, en théorie, la vitesse de transmission à 19.200 Bauds, et la distance entre appareils connectés à 30 mètres (100 pieds) [ZAKS].

Or, on est contraint de connecter chaque appareil au réseau par l'intermédiaire de ces interfaces RS 232. Ces considérations auront donc une lourde répercussion sur les performances d'un réseau basé sur ces interfaces. Il ne faudra donc pas comparer le réseau issu de cette étude avec d'autres réseaux dont les caractéristiques sont très différentes parce qu'ils possèdent des interfaces spécialisés dans ce

but (Voir troisième partie).
On peut citer en particulier COBUS, LISA et ETHERNET dans la famille des réseaux bus, dont les vitesses varient de 100 à 800 KBps, et TROUT de la famille des réseaux en anneau, qui atteint une vitesse de l'ordre de 100 KBps.

Chapitre 2

2. LES ARCHITECTURES ENVISAGEES

Dans la lignée des tendances actuelles, on pourrait envisager deux architectures pour la réalisation du réseau local :

- Réseau en anneau (à insertion de registre),
- Réseau bus (à contention).

Diverses raisons nous limitent à ces deux architectures, notamment :

- Le fait que l'architecture bus est celle qu'on retrouve le plus souvent au niveau commercial (Ethernet, Lisa) ; l'architecture en anneau semble plutôt être un pôle d'attraction en vue d'études réalisées pour la plupart par des universités (Trout, Liu) ;

- Qu'on a pas le temps d'envisager une nouvelle architecture pour notre réalisation ;

- Par les facilités d'implémentation qu'offrent ces deux architectures ;

- Finalement, à cause de leur principe de décentralisation du contrôle, qui influence la fiabilité du réseau.

Ces deux architectures sont présentées ci-dessous en vue d'une comparaison qui permettra de faire un choix.

2.1. Réseau en anneau

L'architecture dont on s'inspirerait est celle d'un anneau à insertion de registre (Voir II.2.6 Réseaux DCLN de Liu).

Principe

Le principe qu'on adopterait serait le suivant :

- Chaque noeud, en l'occurence chaque North-Star, ne connait que son prédecesseur et son successeur immédiats (Adjacents) et agit comme répétiteur de messages, c'est-à-dire qu'il reçoit un message du noeud qui le précède sur l'anneau (i-1) et qu'il le renvoie au noeud qui le suit (i+1).

Certains noeuds ne retransmettent pas les messages : ce sont les noeuds qui reçoivent, après un tour complet, le message qu'ils ont eux-même envoyé.

- Lorsqu'un noeud désire envoyer un message, il aiguille d'abord le réseau vers un buffer pour éviter que des messages arrivant ne soient perdus pendant qu'il dépose le sien sur l'anneau. Dès qu'il a fini son émission, il branche la sortie du buffer sur l'anneau, de sorte que les messages retardés puissent poursuivre leurs chemins.

- Le noeud destinataire prend une copie du message qui lui est destiné mais ne le retire pas de l'anneau ; cependant, il profite de son passage pour positionner un flag qui servira d'acquittement. Le message retourne vers son expéditeur où il sera retiré de l'anneau, par déconnexion du registre d'insertion.

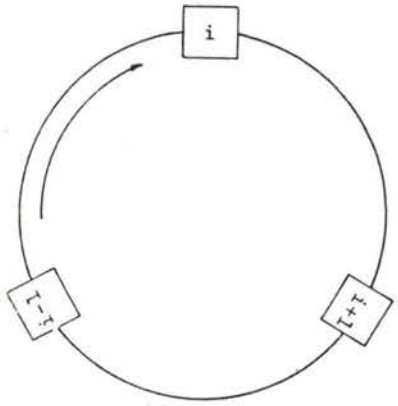


Fig. V.1 Réseau en anneau.

Format des messages

Un message est composé de trois parties :

- L'en-tête qui comprend les adresses d'origine et de destination du message,

- la partie données proprement dite, et

- la queue qui comprend les informations nécessaires au contrôle d'erreur et d'acquittement.

Remarques

1. Chaque porte RS-232 fournit une entrée et une sortie de signaux, ce qui permet d'obtenir un anneau en connectant chaque sortie d'une porte à l'entrée de la porte suivante, jusqu'à fermeture de l'anneau.

2. Chaque noeud doit obligatoirement jouer son rôle de répétiteur sinon l'anneau se brise ; ceci oblige, pour assurer l'intégrité de l'anneau, de prévoir un moyen (Par exemple un relais) qui court-circuite tout noeud inactif.

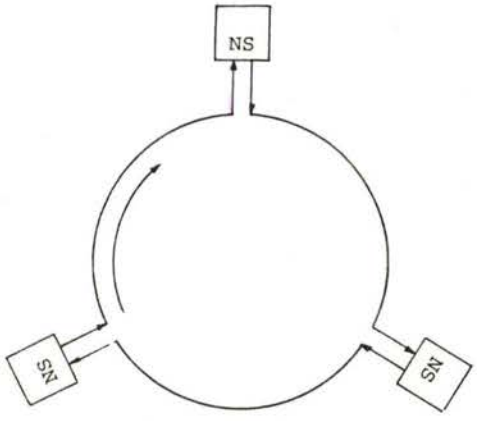


Fig. V.2 L'interconnexion des North-Star en anneau.

2.2. Réseau bus

Il y a plusieurs techniques pour réaliser un réseau bus ; une technique courante, employée par exemple dans Cobus (Lausanne), est la suivante :

Principe

Les North-Star sont tous connectés sur le même câble ;

- Avant d'envoyer son message, un noeud vérifie d'abord (Listen before talk) que le bus est libre, i.e. si aucun autre noeud n'émet de message à ce moment-là ; s'il l'est, le noeud envoie aussitôt son message, sinon il attend que le bus redevienne libre.

- Les autres noeuds écoutent le bus, s'aperçoivent qu'un nouveau message vient de commencer et vérifient chacun s'il leur est destiné ; si oui, le noeud destinataire écoute l'entièreté du message à la suite duquel il envoie, sans perdre le bus, un acquittement au noeud émetteur. Les

noeuds non concernés par ce message-ci attendent le prochain, et ainsi de suite jusqu'à ce qu'ils soient destinataires d'un message émis ou qu'ils veuillent eux-mêmes en émettre un.

- Il n'est cependant pas exclu que deux noeuds, ou plus, désireux d'émettre, s'aperçoivent en même temps que le bus est libre et commencent dès lors en même temps leurs émissions, de sorte que les messages envoyés entrent en collision.
  Pour remédier à ce problème, tout noeud qui émet écoute aussi le bus (Listen while talking) pour vérifier que ce qu'il a envoyé n'a pas été altéré par d'autres messages. Les noeuds qui détectent une altération de leur message, c'est-à-dire une collision, arrêtent aussitôt leurs émissions et attendent pendant un intervalle de temps différent pour chacun et aléatoire avant de reprendre leur envoi.
  Les noeuds destinataires remarquent aussi que le message envoyé est trop court par rapport à un message valide, et attendent donc le prochain message qui leur est destiné.

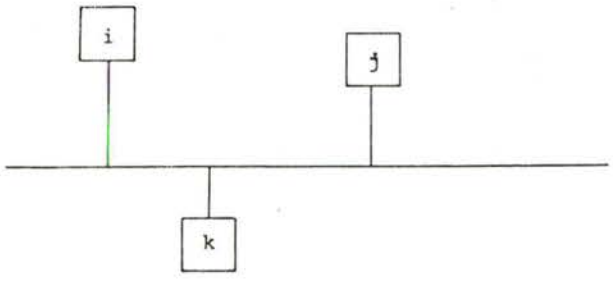


Fig. V.3 Réseau bus.

Format des messages

A tout message entièrement envoyé correspond un acquittement. L'ensemble constitué d'un message et de son acquittement constitue une transaction, tout au long de laquelle le bus apparait occupé à d'autres noeuds qui voudraient émettre.

Tout comme dans l'architecture d'un réseau en anneau, le message est composé de trois parties :

- l'en-tête qui comprend les adresses d'origine et de destination;
- la partie données proprement dite;
- la queue qui comprend les informations nécessaires au contrôle d'erreurs.

L'acquittement est composé d'un ou de deux caractères de contrôle. Il est envoyé dès la fin de réception du message, de sorte que le bus reste toujours occupé, et circule dans le sens inverse du message, c'est-à-dire depuis le destinataire du message vers l'expéditeur de ce même message.

Remarques

1. L'interconnexion des North-Star nécessite un minimum de hardware pour réaliser la connexion au réseau ; ce hardware, ou interface, devrait permettre :

   - de connaître l'état du bus (libre ou occupé), pour réaliser la fonction "Listen before talk".

   - de détecter les collisions, pour réaliser la fonction "Listen while talking" ,

   - d'envoyer et de recevoir des données sur un câble unique (Coaxial) ,

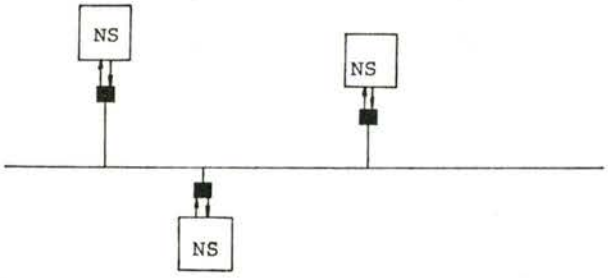


Fig. V.4 Connexion des North-Star au bus via un interface.

2. Il n'y a plus de contraintes sur l'activité des noeuds dans cette architecture-ci.

2.3. Comparaison des architectures

Les besoins immédiats d'un réseau local (Notamment pour l'avancement d'autres mémoires) nous obligent à choisir parmi les deux architectures que l'on vient d'exposer. Le choix ne se fera pas sur base de performances espérées de chacune de ces architectures, qui seront en tous les cas limitées du fait de l'utilisation des portes RS-232. Au contraire, le choix se fera sur base des deux critères suivants :

- Facilité de réalisation matérielle ;
- Fiabilité du réseau.

2.3.1. Avantages et inconvénients des architectures proposées

Chacune des architectures proposées présente certains avantages et inconvénients, au niveau des critères cités, qu'il est utile de spécifier :

Réseau en anneau :

Avantage :

L'avantage d'une telle solution réside dans sa facilité de réalisation puisque cette architecture ne requiert aucun matériel supplémentaire ; les portes RS-232 peuvent en effet être utilisées telles quelles.

Inconvénient :

L'inconvénient majeur de cette solution provient de la (trop) grande dépendance des noeuds les uns les autres, par le fait qu'un seul d'entre eux provoque des troubles au niveau du réseau s'il ne peut subitement plus assurer sa fonction de répétiteur.

Réseau bus :

Avantage :

L'un des principaux avantages qu'offre le réseau bus provient de l'indépendance des noeuds qui, s'ils deviennent inactifs, ne provoquent que peu de trouble au niveau du réseau.

Inconvénient :

Néanmoins, cette architecture nécessite un interface spécialisé qui permet de connecter plusieurs stations sur un même bus (Câble coaxial).

2.3.2. Choix d'une architecture

Le critère de la fiabilité, exige que le comportement anormal (Arrêt, Re-boot, ...) d'un noeud n'affecte pas, ou du moins très peu, celui du réseau. Pareil critère écarte tout de suite l'architecture en anneau.

Pour ma part, la nécessité de réaliser un interface pour l'architecture du type bus constitue un attrait nouveau.

Toutefois, on reportera au maximum des décisions d'architecture au niveau software pour limiter nos problèmes et nos incompétences qui surgiraient si de telles décisions devaient être prises au niveau hardware.

L'architecture retenue est donc bien évidemment celle du réseau bus.

Chapitre 3

3. REALISATION D'UN RESEAU BUS

3.1. Spécifications fonctionnelles du réseau bus

Les spécifications fonctionnelles du réseau concernent son principe de fonctionnement, ses caractéristiques essentielles et ses objectifs.

3.1.1. Principe du réseau bus

Le principe du réseau bus qu'on se propose de réaliser a déjà été abordé au point 2.2 de cette partie.

3.1.2. Caractéristiques essentielles

Les caractéristiques essentielles de ce réseau sont :

- Nombre maximum de stations : 255.
- Support de transmission utilisé : paire blindée de câbles torsadés.
- Type de transmission : Standard RS-232 (Avis V24 du CCITT).
- Taux de transfert : 4.800 Bps.
- Longueur maximum du réseau : A déterminer.
- Procédure de contrôle des communications : basée sur un concept multi-accès sans "convention" avec détection et résolution de collision (Carrier Sense, Multiple Access with collision detection : CSMA/CD).
- Longueur des messages échangés entre processus d'application : longueur variable mais limitée à 250 octets.

3.1.3. Objectifs recherchés

Les objectifs recherchés dans la conception du réseau se résument aux propriétés suivantes :

- Simplicité : tous les dispositifs tendant à compliquer la construction et le fonctionnement du réseau ont été écartés

de la conception ;

- Compatibilité : la compatibilité vise la possibilité d'interconnexion de systèmes issus d'implémentations diverses ; ces systèmes doivent toutefois être compatibles avec l'interface de connexion au réseau, et avec le protocole du niveau application des autres systèmes du réseau avec lesquels ils veulent communiquer ;

- Equité : tous les noeuds disposent de la même priorité d'utilisation du réseau ;

- Immobilisation : aucun noeud ne peut immobiliser à lui seul les ressources du réseau, empêchant ainsi les autres noeuds de communiquer ;

- Stabilité : l'accroissement du temps d'attente doit être proportionnel par rapport à l'accroissement de la charge sur le réseau ;

- Fiabilité et maintenabilité : un des objectifs fondamentaux exige du réseau qu'il soit de fiabilité maximum et de maintenance facile ;

- Architecture en couches : une architecture en couches a été adoptée afin d'obtenir une division claire et nette de la conception de ce réseau selon des principes qui tendent de plus en plus à devenir des standards. Cette architecture en couches a également pour avantage de simplifier les spécifications.

3.1.4. Remarques

Pour des raisons de simplicité et d'efficacité dans la conception du réseau, un certain nombre d'options ont été délibérément écartées :

- Vitesse : on ne recherche pas les performances au niveau vitesse de transmission, ces performances exigeant généralement des équipements matériels spécialisés ;
  Tous les systèmes connectés doivent se mettre d'accord sur l'utilisation d'une vitesse de transmission commune (Par exemple 4.800 Bps) ;

- Sécurité : les couches de contrôle du réseau ne procèdent à aucune opération particulière de confidentialisation de l'information ; aucune protection du réseau contre une tentative de connexion irrégulière n'est assurée.

3.2. Architecture en couches

Sans vouloir faire explicitement référence au découpage proposé par l'ISO, il est évident que toute conception saine d'un système de transmission se base sur une architecture en couches. La terminologie utilisée ne fait donc nullement référence à celle utilisée par l'ISO, bien qu'il existe des similitudes entre elles.
L'architecture se décompose en quatre couches :

- La couche application,
- La couche transport,
- La couche liaison de données,
- La couche physique.

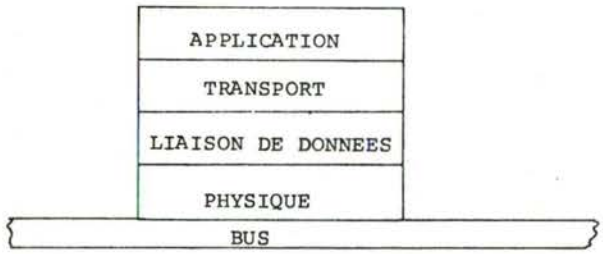


Fig. V.5 Les quatre couches de l'architecture.

Chaque couche de l'architecture est décrite en terme de fonctions qu'elle réalise et en terme de services qu'elle procure à la couche directement supérieure. Ce principe, directement inspiré des travaux de l'ISO [1], permet ainsi d'obtenir une démarche d'analyse "Top-Down" qui, en partant des besoins globaux qu'ont les processus d'application (PA) d'un réseau local, aboutit finalement à la description des fonctions élémentaires nécessaires que doivent réaliser les trois couches inférieures de l'architecture (Transport, Liaison de données, Physique).

Les requêtes de services sont exprimées sous forme d'appels à des procédures Pascal ; ceci permet de simplifier et de clarifier le modèle proposé [2].

3.2.1. La couche application

On ne décrira pas les fonctions de la couche application mais les services qu'elle désire obtenir en vue d'une communication inter-processus.

---

[1] : Voir 3ème Partie de ce mémoire

[2] : Un modèle procédural exprimé uniquement en termes de procédures figure dans les annexes [Annexe C1]

3.2.1.1. Les besoins globaux des processus d'application

Les besoins d'utilisation d'un réseau local sont encore mal définis. Pour l'instant ils se limiteraient à l'échange d'informations entre les différents processus d'application (PA) qui s'exécutent dans les stations connectées au réseau.

On entend par échange d'informations :

- L'envoi de messages à destination d'un autre PA ;
- La réception de messages en provenance d'un autre PA.

Un message est une suite d'octets. Aucune restriction n'est faite quant au contenu de ces octets : ce peut être, par exemple, des données binaires en provenance d'un fichier objet ou des caractères en provenance d'un fichier de texte. Pour des raisons de facilité d'implémentation, la longueur d'un message est variable mais limitée à 250 octets. Cette longueur est aussi un compromis entre la taille des informations à transmettre sur le bus et le temps de réservation du bus pour ce transfert.

Toute station est identifiée de manière unique par une adresse. Cette adresse lui permet notamment de reconnaitre un message qui lui est destiné.
Pour des raisons de facilité d'implémentation et de simplicité des traitements à effectuer, l'adresse est constituée d'un nombre, compris entre 1 et 255, plutôt que d'un nom logique.
Puisqu'il n'y a jamais qu'un seul PA qui s'exécute à la fois dans chaque station, en adressant celle-ci on adresse aussi le PA qui s'y déroule.

3.2.1.2. Les services requis

Les services requis par un PA, en vue de l'échange d'informations avec d'autres PA, doivent lui permettre :

- D'initialiser l'accès au réseau ;
- De recevoir des messages en provenance de n'importe quelle station du réseau ,i.e. de n'importe quel PA ;
- D'envoyer des messages à n'importe quel PA du réseau ;
- De cloturer l'accès au réseau lorsque le PA n'a plus besoin d'y accéder.

Ces quatres services sont fournis au PA sous forme d'appel à des procédures Pascal. Voici les spécifications de ces procédures :

1. OPEN_LINK (VAR host_addr:address; VAR err:code)

   Demande d'initialisation d'une liaison virtuelle.

Cette procédure permet au PA d'accéder au réseau. Elle renvoie au PA appelant l'adresse de la station (host_addr) et un code d'erreur eventuel (err), par exemple :

- Accès déjà ouvert ;
- Autre station déjà connectée sous la même adresse.

2. CLOSE_LINK (VAR err:code)

Demande de clotûre

Cette procédure a pour but de fermer l'accès au réseau.
Après l'appel à cette procédure, la station est déconnectée du réseau de sorte que le PA ne puisse plus recevoir de messages en provenance des autres PA qui accèdent au réseau.
Un code d'erreur (err) est retourné au PA, indiquant les conditions de réalisation de cette procédure :

- Accès déjà fermé ;
- Messages reçus mais non retirés par le PA.

3. SEND (to_addr:address; len:length; msg:message; VAR err:code)

Demande d'envoi d'un message.

Cette procédure a pour but d'envoyer un message (msg) dont on connait la longueur (len) à la station dont on spécifie l'adresse (to_addr).
Pour des raisons diverses, il se peut que le destinataire n'ait pas reçu le message ou l'ait refusé ; la procédure d'envoi applique la politique du meilleur effort pour assurer un service sûr et fiable au PA utilisateur. Le PA peut savoir dans quelles conditions l'envoi s'est réalisé ; dans ce but, la procédure renvoie au PA un code d'erreur (err), par exemple :

- Trop d'essais ;
- Station destinatrice inexistante ;

4. RECEIVE (VAR from_addr:address; VAR len:length; VAR msg:message; wait:flag, VAR err:code)

Demande de réception d'un message.

Deux solutions sont possibles lors de l'appel à cette procédure. Dans la première solution (wait=vrai), un message peut être déjà disponible,

auquel cas la procédure renvoie ce message (msg), sa longueur (len) et l'adresse de son expéditeur (from_addr). Si aucun message n'est disponible lors de l'appel à cette procédure, le PA est bloqué jusqu'à disponibilité d'un nouveau message. Dans la seconde solution (wait=faux), si aucun message n'est disponible, le PA n'est pas bloqué et un code d'erreur (err) est renvoyé. Si un message est disponible, il est fourni au PA comme dans le cas précédent.

N.B. : Un PA peut faire appel à cette procédure, même après clotûre de l'accès au réseau, pour prendre les messages restant qui ont été reçus avant cette clotûre.

3.2.1.3. Synchronisme et asynchronisme des services

Certaines exigences sont faites quant à la fiabilité de réalisation de ces services.

On souhaite qu'un message soit correctement reçu par le destinataire. Si l'envoi n'a pu s'effectuer dans les conditions espérées (A la suite de plusieurs essais, le message n'a toujours pas été correctement reçu ou n'a pas été du tout reçu à cause de mauvaises conditions de transmissions, parce que la station destinatrice est absente, ou parce que le réseau est fort chargé et que le message est entré en collision avec d'autres messages à plusieurs reprises.) on désire que le PA en soit averti pour qu'il puisse prendre des mesures appropriées.

Ces considérations imposent à la procédure d'envoi (SEND) qu'elle se déroule de façon synchrone par rapport au PA puisqu'il doit, en effet, savoir dans quelles conditions le services d'envoi s'est terminé.

Cette fiabilité, au niveau de l'expédition de messages, exige des stations connectées au réseau qu'elles puissent, en permanence, recevoir des messages qui leur sont destinés. Il est souhaitable, dès lors, que ces messages reçus soient aussitôt stockés et que le PA émetteur soit averti de leur bonne réception.
Le stockage des messages reçus se fait dans un tampon -appelé panier- prévu à cet effet, jusqu'à concurrence de l'espace disponible.
Les messages reçus et stockés dans le panier peuvent ensuite être délivrés au PA au fur et à mesure que celui-ci en fait la demande (RECEIVE), jusqu'à concurrence du nombre de messages disponibles.

Le mécanisme de réception et de stockage que l'on vient d'expliquer, et par lequel des messages peuvent être reçus et stockés sans l'intervention du PA, se déroule de façon asynchrone par rapport à l'exécution de ce PA.

3.2.2. La couche transport

De manière générale, la couche transport permet un transfert de données entre PA qui accèdent au réseau (De bout en bout puisqu'une station ne supporte qu'un seul PA à la fois).

La couche transport décharge ses utilisateurs, les PA, de tous les détails liés au transfert de données sûr et à coûts réduits.

3.2.2.1. Services fournis

La couche transport fournit un ensemble de services à destination de la couche supérieure. Ces services sont les services demandés par la couche application :

- Ouverture de l'accès au réseau (OPEN_LINK) ;
- Envoi de messages à destination d'une autre station (SEND) ;
- Réception de messages en provenance d'une autre station (RECEIVE) ;
- Fermeture de l'accès au réseau (CLOSE_LINK) .

3.2.2.2. Définitions des éléments fonctionnels

Il est nécessaire de définir les éléments qui sont le support des traitements effectués par la couche transport. Les termes définis sont :

Trame : une trame est le regroupement d'informations fournies par le PA (Adresse du destinataire, message à envoyer) et d'informations de contrôle nécessaires au protocole propre de la couche transport.
Une trame est constituée de trois parties distinctes :

- L'en-tête, qui comprend des informations utiles au transfert de données : drapeau de début de trame, adresse d'origine et de destination, type du message, longueur du message.
- Le message de données lui-même.
- La queue, qui comprend les informations nécessaires au contrôle d'erreurs et le drapeau de fin de trame.

La figure V.6 donne le format d'une trame.

Acquittement : un acquittement, associé à une trame, est constitué d'une seule information. Il permet à l'expéditeur d'une trame, c'est-à-dire la couche transport, de savoir dans quelles conditions cette trame a été reçue par le destinataire.
Pour refléter toutes les conditions possibles,

l'acquittement peut prendre différentes valeurs :

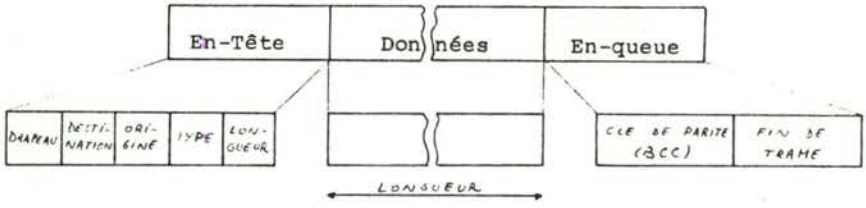


Fig. V.6 Format d'une trame.

- "ACK" : la trame est correctement reçue et est acceptée par le destinataire.
- "NAK" : la trame est mal reçue et est donc refusée par le destinataire.
- "DLE" : la trame est correctement reçue mais est refusée par le destinataire.
- "NUL" : aucun acquittement ne fait suite à l'émission d'une trame.
- "CAN" : aucun acquittement n'est attendu étant donné que la trame est entrée en collision avec une autre.

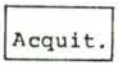


Fig. V.7 Format d'un acquittement.

Transaction : une transaction est l'ensemble formé d'une trame et de l'acquittement associé à cette trame (Fig V.8)

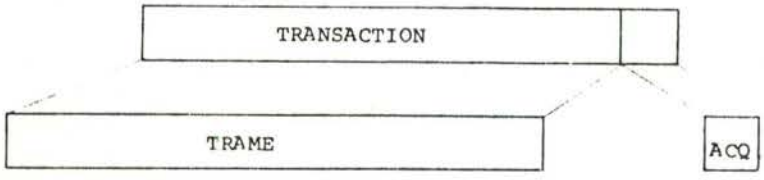


Fig. V.8 Format d'une transaction.

3.2.2.3. Fonctions réalisées

Pour être capable de fournir les services précités, la couche transport doit réaliser un certain nombre de fonctions.

On distingue deux catégories de fonctions à réaliser :

- Les fonctions propres à l'émission de messages à destination d'une autre station : on les appellera les fonctions d'émission.

- Les fonctions propres à la réception de messages en provenance d'une autre station : on les appellera les fonctions de réception.

A) Les fonctions d'émission :

1. Préparation d'une trame : la préparation d'une trame consiste dans l'encadrage du message de données par les informations de contrôle adéquates. C'est l'encapsulage (Voir figure V.6).

2. Gestion des transactions : la couche transport se charge, dans le but d'offrir un service sûr et fiable aux utilisateurs, de gérer les transactions. Cette gestion se fait en deux étapes successives :

   1. Demande d'envoi de la trame préparée,
   2. Réception d'un acquittement associé à cette trame.

   Chaque étape est réalisée sous forme de requête à un service de la couche inférieure.

   La gestion des transactions consiste à effectuer les opérations nécessaires pour assurer la transmission parfaite d'une trame.
   Cette transmission peut soit avoir abouti correctement, soit incorrectement ou même pas du tout.

   - Transmission correcte : la station destinatrice a correctement reçu la trame et l'a soit acceptée, soit refusée par manque de place dans son panier ;

   - Transmission incorrecte : la station destinatrice a reçu la trame, mais celle-ci était incorrecte (Vérification de la parité longitudinale) ;

   - Transmission non aboutie : la transmission peut ne pas avoir abouti du tout soit parce que la trame est entrée en collision avec une autre, soit parce que la station destinatrice n'est pas connectée au réseau.

   Tant que la transmission n'a pas abouti correctement, la fonction de gestion des transactions

effectue de nouveaux essais. Cependant, le nombre d'essais est limité, pour ne pas surcharger le réseau. Actuellement, la limite est fixée à 5 essais pour chaque cas de non aboutissement (Collision, Acquittement négatif ou non présent) et à 10 essais en tout.

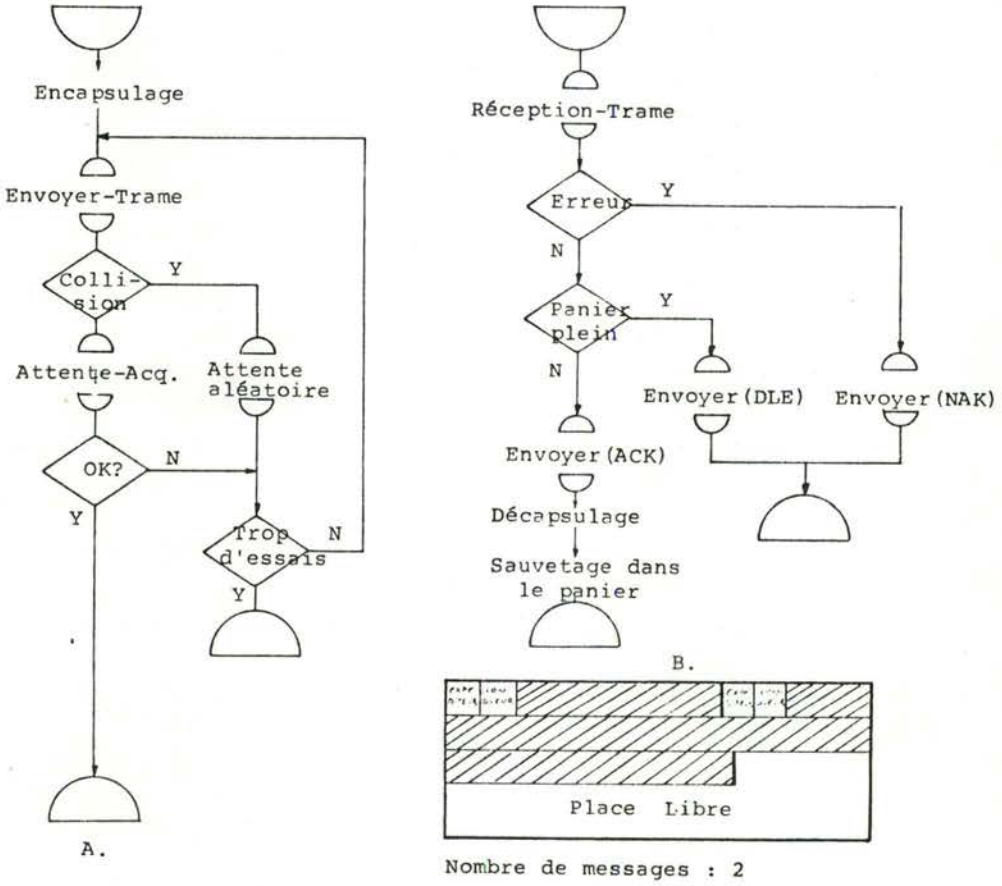


Fig. V.9 a&b Diagramme d'enchainement.

En résumé, les fonctions d'émission reprennent :

- L'encapsulage des trames ;
- La gestion des transactions : retransmission un certain nombre de fois tant que la transmission d'une trame n'a pas correctement abouti.

La figure V.9a donne le diagramme d'enchainement de ces fonctions.

B) Les fonctions de réception :

1. Gestion des transactions : la gestion des transactions se fait en deux étapes :

1. Réception d'une trame : la couche liaison de données ne délivre que les trames complètes et à destination de la station. Dans le but d'envoyer un acquittement à la station émettrice, la couche transport détermine si la trame a été correctement reçue ou non (Par contrôle de la clé de parité transversale), et si le panier peut recevoir le message de données.
2. Demande d'envoi de l'acquittement : la couche transport détermine l'acquittement à envoyer (ACK, NAK ou DLE) et fourni cet acquittement à la couche inférieure pour envoi.

2. Sauvetage des messages : finalement, pour autant que la trame ait été correctement reçue et que le panier ne soit pas trop rempli, la couche transport extrait le message de la trame (Décapsulage) en vue de son stockage.

En résumé, les fonctions de réception reprennent :

- La gestion des transactions : détermination de l'acquittement ;
- Le décapsulage d'une trame ;
- La gestion du panier ;
- Le stockage du message.

La figure V.9b donne le diagramme d'enchainement de ces fonctions.

3.2.2.4. Services requis

La couche liaison de données doit être en mesure de fournir les services suivants à la couche transport :

- Envoi de trames ;
- Réception de trames ;
- Envoi d'acquittement ;
- Réception d'acquittement.

3.2.3. La couche liaison de données

La couche liaison de données contrôle les transmissions de trames et d'acquittements, en détectant notamment les erreurs qui pourraient survenir au niveau physique de l'architecture.

3.2.3.1. Services fournis

La couche liaison de données fournit quatre services à destination de la couche supérieure. Pour rappel, ces

services sont :

- Envoi de trames sur le réseau ;
- Réception de trames adressées à la station ;
- Envoi d'acquittements associés aux trames reçues ;
- Réception d'acquittements associés aux trames envoyées.

3.2.3.2. Définition des éléments fonctionnels

Il est nécessaire de définir les éléments qui sont le support des traitements effectués par la couche liaison de données.

Bytes : les trames et les acquittements manipulés par la couche transport sont subdivisés en composants appelés bytes (Suite de bits).

3.2.3.3. Fonctions réalisées

Tout comme dans la couche transport, on distingue deux catégories de fonctions à réaliser : les fonctions d'émission et les fonctions de réception.

A) Les fonctions d'émission

1. Désassemblage d'une trame : la trame est désassemblée en bytes en vue de sa transmission ; les trames sont, à ce niveau-ci, vues comme une suite de bytes.
   Puisque le drapeau (Caractère BEL de l'alphabet international no. 5) est utilisé pour distinguer le début d'une trame, il faut éviter qu'il n'apparaisse parmi les autres bytes de cette même trame. Dans ce but, tout byte autre que le drapeau qui a la valeur BEL, est dédoublé de la manière suivante :

   BEL ---> DLE PAD.

   Le caractère DLE devient alors un caractère de contrôle qu'il convient aussi de dédoubler lorsqu'il apparait parmi les bytes d'une trame :

   DLE ---> DLE DLE.

2. Envoi de bytes : chaque byte de la trame ainsi désassemblée est fourni à la couche inférieure pour envoi.

3. Envoi d'acquittement : l'acquittement déterminé par la couche transport doit être envoyé directement après avoir reçu le dernier byte de la trame pour ne pas perdre le bus.

4. Détection du bus libre : (Listen before talk) avant

d'expédier le premier byte d'une trame, il est nécessaire de s'assurer qu'aucune autre transmission n'est en cours ; si c'était le cas, il faudrait attendre que le bus redevienne libre.
Cette fonction permettra aussi de s'apercevoir de l'absence d'acquittement. Celui-ci doit en effet parvenir dans un laps de temps relativement court après le dernier byte de la trame pour éviter de perdre le bus pendant une transaction.

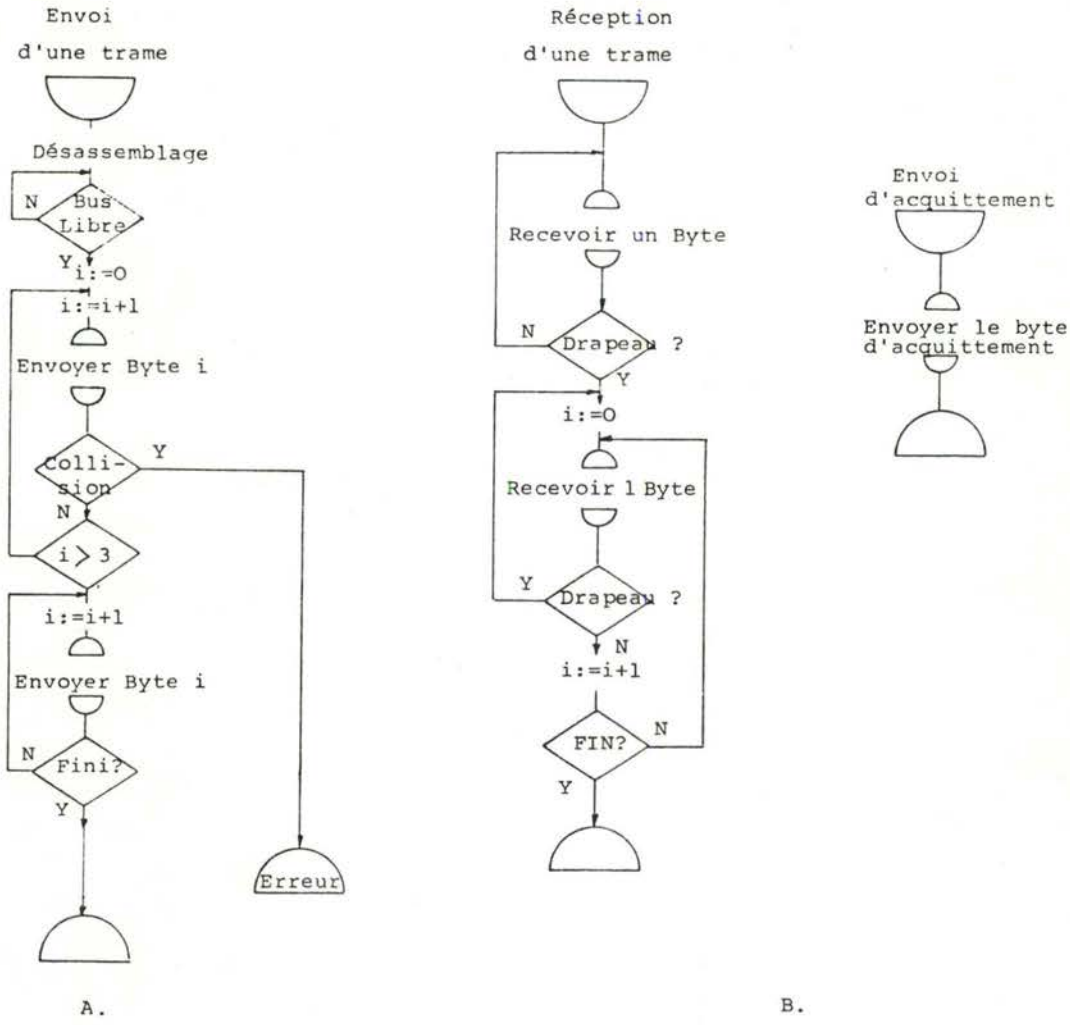


Fig. V.10 a&b Diagramme d'enchainement.

5. Détection des collisions : (Listen while talking) une collision ne pouvant survenir qu'au début de l'expédition d'une trame (Deux stations découvrant le bus libre en même temps), il est indispensable de s'assurer que les bytes expédiés ne sont pas altérés par d'autres expéditions.
La détection de collision est effectuée sur les trois premiers bytes d'une trame (Fig. V.6 : Drapeau, Destination et Origine). En effet , le premier byte d'une trame est le même pour toutes les trames (Drapeau=BEL) ; le deuxième byte (Adresse du destinataire) peut être le même si deux PA décident

d'envoyer leur message au même destinataire ; le troisième byte (Adresse de l'expéditeur) est différent pour toutes les trames envoyées simultanément puisqu'une station ne peut envoyer qu'une seule trame à la fois.
Si une collision est détectée, l'expédition de la trame est aussitôt stoppée et le niveau transport est averti de l'avortement du service (CAN).

En résumé, les fonctions d'émissions reprennent :

- Le désassemblage d'une trame en bytes ;
- L'envoi de bytes ;
- L'envoi d'acquittement ;
- La détection du bus libre ;
- La détection des collisions.

La figure V.10a donne le diagramme d'enchainement de ces fonctions

B) Les fonctions de réception

1. Réception de bytes : chaque byte reçu est fourni par la couche inférieure en vue d'être traité.

2. Détection d'une nouvelle trame : le drapeau ne pouvant figurer qu'au début d'une trame, son apparition provoque aussitôt la synchronisation du traitement sur la trame qui apparaît.
La trame n'est traitée que si elle est destinée à la station, sinon on attend l'arrivée d'une nouvelle trame.

3. Détection des erreurs de transmission : une clé de parité transversale est élaborée sur tous les bytes d'une trame

4. Assemblage des trames : reconstitution d'une trame complète sur base des bytes reçus.
Ce n'est que lorsqu'elle est complètement reçue que la trame est transmise à la couche transport.
N.B. : L'apparition du drapeau en cours d'assemblage d'une trame, provoque aussitôt l'abandon de la trame en cours d'assemblage au profit de celle qui arrive ; c'est généralement le cas des trames entrées en collision.

5. Attente d'acquittement : La couche liaison de données attend un acquittement pendant un laps de temps déterminé ; il est en effet possible qu'aucun acquittement ne fasse suite à l'envoi d'une trame (NUL). Si l'acquittement est reçu il est transmis tel quel à la couche supérieure.

En résumé, les fonctions relatives à la réception de trames

reprennent :

- La réception de bytes ;
- La détection d'une nouvelle trame ;
- La détection des erreurs de transmissions ;
- L'assemblage d'une trame ;
- L'attente et l'envoi d'acquittement.

La figure V.10b donne le diagramme d'enchainement de ces fonctions.

3.2.3.4. Services requis

La couche physique doit être en mesure de fournir les services suivants à la couche liaison de données :

- Envoi de bytes sur le réseau ;
- Réception de bytes par le réseau ;
- Détermination de l'état du réseau.

Le mécanisme de détection de collision (Listen while talking) exige en outre que l'émission et la réception de bytes soient deux processus qui puissent s'exécuter simultanément.

3.2.4. La couche physique

La couche physique fournit les moyens électriques et procéduraux pour la transmission de bytes entre couches de liaison de données qui communiquent.

3.2.4.1. Services fournis

La couche physique fournit trois services à destination de la couche supérieure. Pour rappel, ces services sont :

- Emission de bytes sur le réseau ;
- Réception de bytes en provenance du réseau ;
- Détermination de l'état du réseau.

3.2.4.2. Définition des éléments fonctionnels

Il est nécessaire de définir les éléments qui sont le support des traitements effectués par la couche physique.

Bit : les bytes échangés avec la couche liaison de données, sont subdivisés en bits d'information en vue de leur transmission sur un moyen physique. Un byte

est constitué de 8 bits d'information auxquels s'ajoute un bit de parité.

3.2.4.3. Fonctions réalisées

Comme dans les couches précédentes, on distingue deux catégories de fonctions : les fonctions d'émission et les fonctions de réception.

A) Les fonctions d'émission

1. Préparation du byte : le byte de donnée transmis par la couche liaison de données est divisé en bits d'information susceptibles d'être transmis.
   Une clé de parité, nécessaire pour le contrôle d'erreurs propre à la couche physique, est élaborée sur chaque byte : elle consiste en un bit supplémentaire appelé bit de parité.

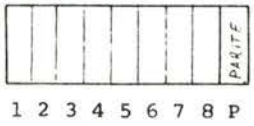


Fig. V.11 Format d'un byte.

2. Envoi des bits sur le bus : la couche physique s'occupe ensuite de l'envoi des bits sur le support physique (Le bus). La transmission s'exécute en mode asynchrone.

En résumé, les fonctions d'émission reprennent :

- Le désassemblage d'un byte en bits ;
- L'élaboration d'une clé de parité ;
- L'envoi des bits sur le bus selon un mode de transmission asynchrone.

La figure V.12a donne le diagramme d'enchainement de ces fonctions.

B) Les fonctions de réception

1. Détection de bits : la couche physique peut détecter le passage de bits sur le support physique (Détection de porteuse sur le bus : "Carrier sense"). Ceci lui permet notamment de connaître l'état du bus (Libre ou occupé).
   Les bits qui circulent sur le bus sont copiés en vue d'un traitement ultérieur.

2. Reconstitution d'un byte : les bits ainsi copiés sont assemblés jusqu'à l'obtention d'un byte. Ce byte sera ensuite tranmis à la couche supérieure. Toute détection d'erreur (Parité, format,..) est signalée à la couche supérieure.

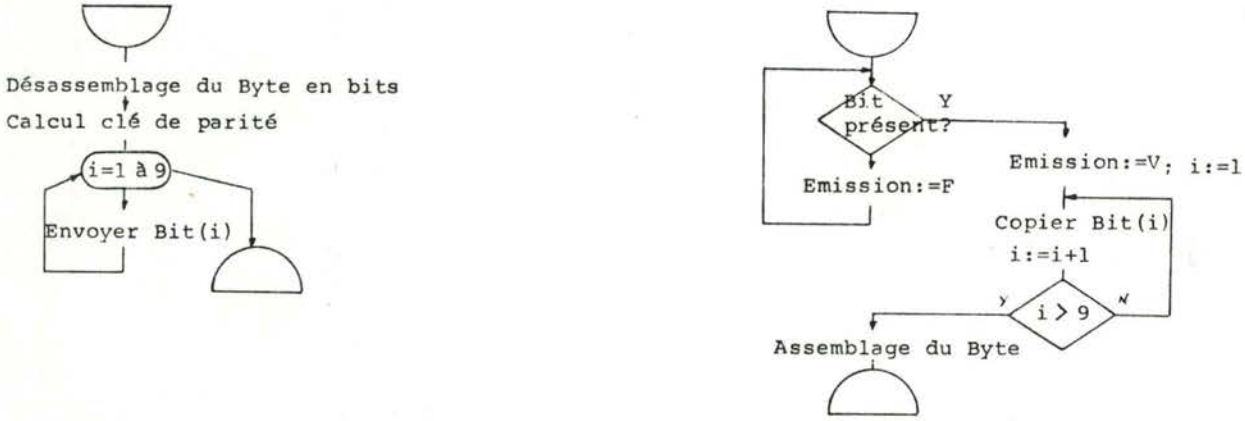


Fig. V.12 a&b Diagramme d'enchainement des fonctions.

En résumé, les fonctions de réception reprennent :

- La détermination de l'état du bus ;
- La détection de bits d'information sur le bus (Carrier Sense) ;
- L'assemblage de bits en byte.

La figure V.12b donne le diagramme d'enchainement de ces fonctions.

C) Remarques : Le mécanisme de détection des collisions (Listen while talking) exige que la réception et l'émission de bytes puisse se faire simultanément. Cette faculté est connue sous le nom de Full duplex.

3.2.5. Conclusions

La figure V.13a reprend, sous forme schématique, l'architecture du modèle retenu. Pour chaque couche, on retrouve les fonctions réalisées, réparties en catégories, et les services qu'elle offre à la couche de niveau supérieur. Les services offerts sont typiquement les points d'interaction entre deux couches adjacentes ; ce sont les interfaces inter-couches.
La figure V.13b reprend les éléments fonctionnels propres au protocole de chaque couche.

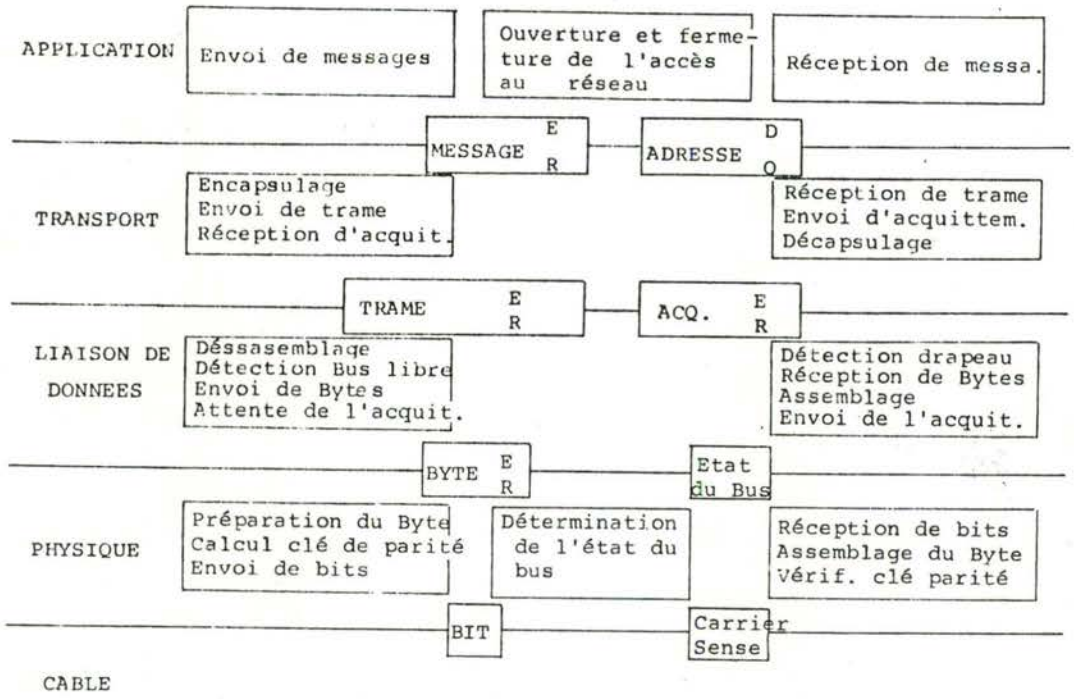


Fig. V.13a Le modèle d'architecture en couches.

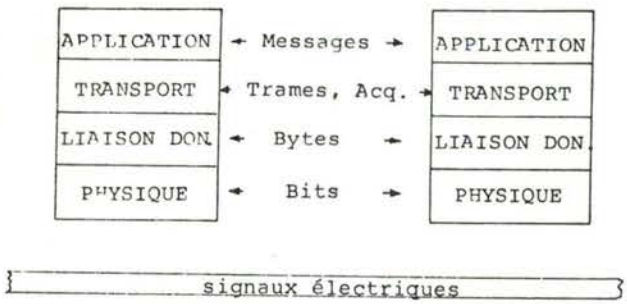
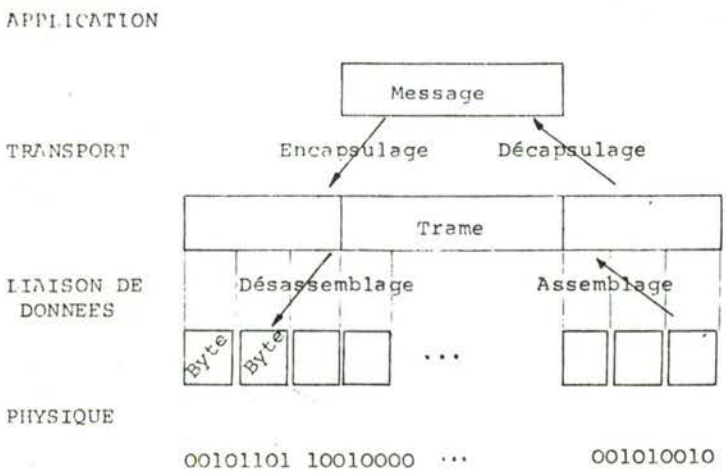


Fig. V.13b Elèments fonctionnels propres aux couches.



3.3. L'interface réseau : le NIU

Ce sont les trois couches inférieures de l'architecture (Transport, Liaison de données et Physique) qui permettent à la couche Application d'accéder au réseau. Elles constituent l'interface-réseau (NIU : Network Interface Unit) qui effectue les opérations nécessaires pour fournir des services sûrs et fiables d'accès au réseau.

Le NIU permet au PA d'accéder au réseau tout comme le gérant des fichiers lui permet d'accéder aux disques (Fig. V.14).

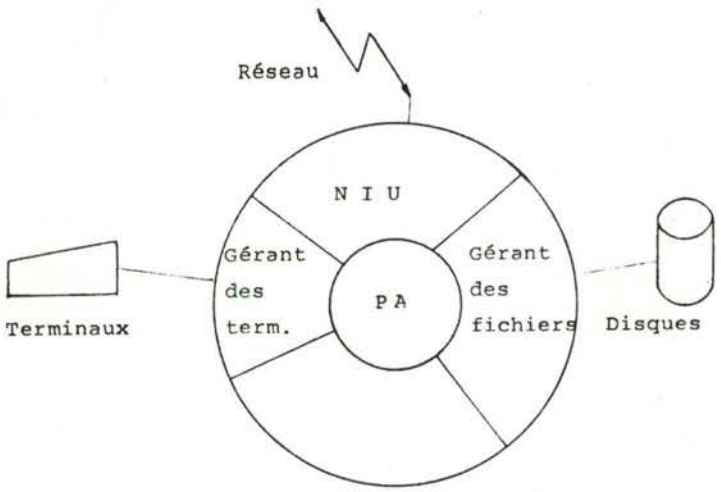


Fig. V.14. Situation du NIU.

3.3.1. Services fournis

Dorénavant, les PA peuvent communiquer entre eux, sur base d'un protocole qui leur est propre, en requérant les services du NIU.
Les services du NIU sont précisemment ceux que la couche transport met à disposition de ses utilisateurs. On y retrouve donc les quatres procédures OPEN_LINK, CLOSE_LINK, SEND, RECEIVE.

3.3.2. Fonctions réalisées

Le but n'est pas ici d'énumérer toutes les fonctions élémentaires réalisées par chacune des couches de l'architecture, mais d'exprimer en quelques mots la fonction globale du NIU.

Le NIU doit être capable, à partir de signaux électriques circulant sur une ligne physique (Le bus), de reconstituter un message valide à l'usage du PA. Inversément, il doit être capable de transformer le message fourni par le PA, en signaux électriques en vue de sa transmission à destination d'un autre

NIU.

3.3.3. L'implémentation

L'implémentation aborde la conception logique et physique du NIU (Software et Hardware).
La conception des couches transport et liaison de données est logique (Software) et fait appel à la notion de programmes ; la conception de la couche physique est physique (Hardware) et fait appel à la notion de composants électroniques.

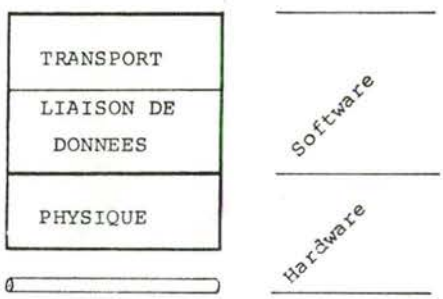


Fig. V.15 Conception du NIU.

A l'exception de l'interface d'adaptation au bus (Voir V.4.4.2), le NIU fait partie intégrante de l'ordinateur. Les programmes, réalisant le protocole d'accès au bus, s'exécutent dans la même mémoire que celle des programmes d'application ; les interfaces qui supportent les standards RS-232 font partie de l'ordinateur-même.

3.4. Conception de la partie hardware du NIU

Les fonctions de la couche physique de l'architecture (Décomposition d'un byte en bits, conversion d'un bit logique en signal électrique, selon les standards RS-232, et inversément) sont réalisées par des circuits électroniques. Ces circuits, généralement intégré en un seul "Chip" qu'on appelle circuit intégré (Integrated circuit : IC), sont connus sous le nom d'interface-série.

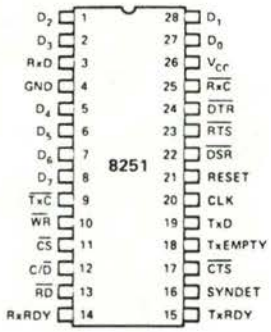
La connexion physique de plusieurs interfaces de ce type sur une seule et même ligne pose un certain nombre de problèmes : pour résoudre ceux-ci, il a fallu développer un interface d'adaptation appelé interface-bus.

3.4.1. L'interface-série RS-232

3.4.1.1. Spécifications de l'interface-série

Dans notre cas, le circuit utilisé par le North-Star pour effectuer les fonctions de sérialisation, est un INTEL 8251. Le lecteur se réfèrera à la quatrième partie de ce mémoire pour avoir de plus amples renseignements sur ce circuit (Voir IV.2.1.2 1er et Annexe D2)

PIN CONFIGURATION



Pin Name	Pin Function
D_7-D_0	Data Bus (8 bits)
C/D	Control or Data is to be Written or Read
$\overline{RD}$	Read Data Command
$\overline{WR}$	Write Data or Control Command
$\overline{CS}$	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for char. from 8080)

Pin Name	Pin Function
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET	Sync Detect
$\overline{RTS}$	Request to Send Data
$\overline{CTS}$	Clear to Send Data
TxE	Transmitter Empty
Vcc	+5 Volt Supply
GND	Ground

Fig. V.16 L'interface 8251.

On peut néanmoins rappeler les fonctions essentielles que ce circuit réalise :

- Conversion de signaux électriques en informations logiques (+12V = "0", -12V = "1") et assemblage d'un byte ;
- Désassemblage d'un byte en bits, conversion de bits d'information en signaux électriques.

3.4.1.2. Principe d'interruption

1. Types d'interruptions : l'interface-série fournit toute une série de signaux qui indiquent la réalisation d'évènements. Ces signaux peuvent être utilisés dans un contexte d'interruption.
   On ne retiendra que deux évènements, auxquels on associe un type d'interruption :

   - RxRDY : Un byte est disponible dans l'interface-série pour être traité.
     Ce signal fournira l'interruption Réception-prête.

   - TxRDY : L'interface-série est prêt à recevoir un nouveau byte à émettre.
     Ce signal fournira l'interruption Emission-prête.

2. L'interruption-réseau : Dans le North-Star, tous les signaux qui, en provenance de l'interface, indiquent la réalisation d'un évènement sont regroupés pour ne fournir qu'un seul groupe d'évènements. C'est à ce groupe d'évènements qu'on associe une interruption, appelée Interruption-réseau (NetRDY).

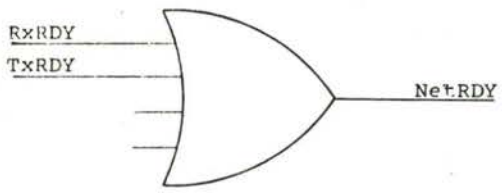


Fig. V.17 Regroupement des évènements.

3. Le gérant de l'interruption-réseau : Lorsque l'interruption-réseau apparaît, le processus en cours (En fait un PA) est interrompu et le gérant des interruptions est activé.

C'est le gérant des interruptions qui détermine, par lecture de l'état de l'interface-série, quel évènement s'est produit et donc quel service d'interruption doit être exécuté.

Une question subsiste : comment le gérant des interruptions est-il activé lors de l'apparition de l'interruption-réseau ?

La présence d'une interruption provoque le dépot d'une instruction sur le bus interne des données ; cette instruction particulière a pour but de brancher, après l'avoir sauvé, le compteur de programme à une adresse définie en mémoire.
A cette adresse se trouve une séquence d'instructions qui permet d'activer le gérant. Dans le cadre du North-Star, il est possible de fixer (Par câblage) cette adresse par groupe d'interruption. La figure V.18 donne un exemple de configuration de la mémoire en ce qui concerne le gérant de l'interruption-réseau.

3.4.2. L'interface-bus

3.4.2.1. Les standards RS-232

Les standards RS-232 définissent les signaux fournis par les interfaces-série auxquels ils se rapportent. Ces signaux concernent principalement la jonction entre équipement terminal de traitement de données (ETTD) et

équipement terminal de circuit de données (ETCD) (Avis V24 du CCITT [MACCHI pg 109]).

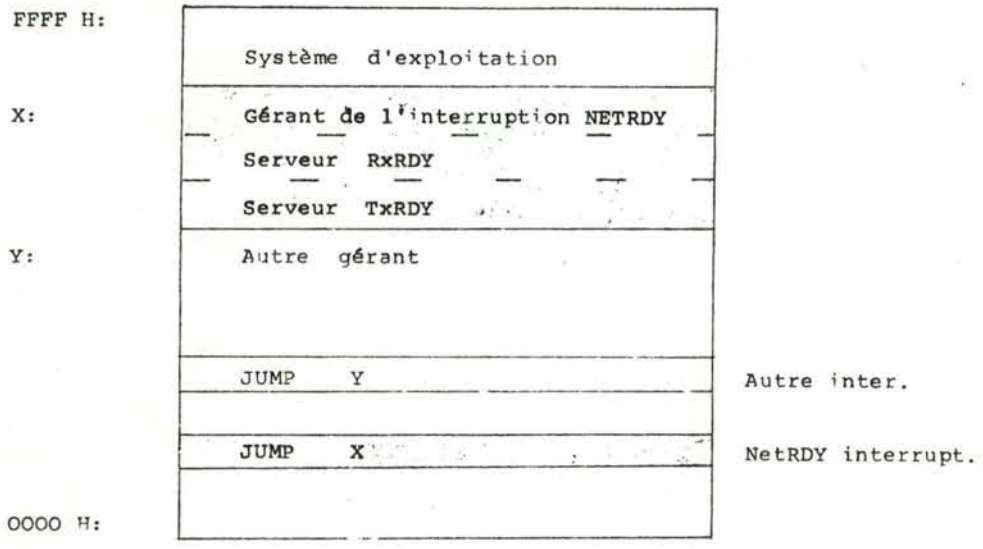


Fig. V.18 Configuration de la mémoire.

Les caractéristiques essentielles du standard RS-232 qui nous intéressent sont les suivantes :

- L'émission et la réception des données se fait sur deux lignes physiques distinctes.
- Une tension positive (+12V) représente un "0" et une tension négative (-12V) un "1".

3.4.2.2. Spécifications de l'interface-bus

Il a fallu résoudre deux problèmes :

- L'émission et la réception des données doit se faire au moyen d'un bus unique : il faudrait donc pouvoir court-circuiter les deux lignes physiques d'émission et de réception ;

- Puisque toutes les stations sont connectées sur un seul bus, il faut éviter les problèmes tels qu'une station émet un "0" (+12V) pendant qu'une autre émet un "1" (-12V) : de telles situations pourraient avoir de graves conséquences sur la vie des circuits électroniques.

L'interface présenté en figure V.19 permet de résoudre ces problèmes.

- De cette manière, la réception et l'émission s'effectuent sur la même ligne ; cette facilité sera également utilisée pour détecter les collisions.

- Il n'y a plus de concurrence quant à la tension déposée par chaque station sur le bus. A l'état de repos, grâce aux deux résistances de bout de ligne, la tension du bus est négative (-12V), ce qui correspond à des "1". D'autre part, la diode ne laisse passer qu'une tension positive, de sorte que les stations ne fournissent plus que les "0" (Les "1" sont fournis par les résistances de bout de ligne).

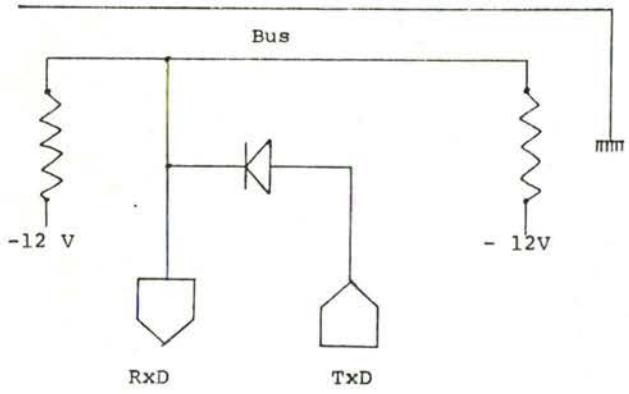


Fig. V.19 Prototype d'interface.

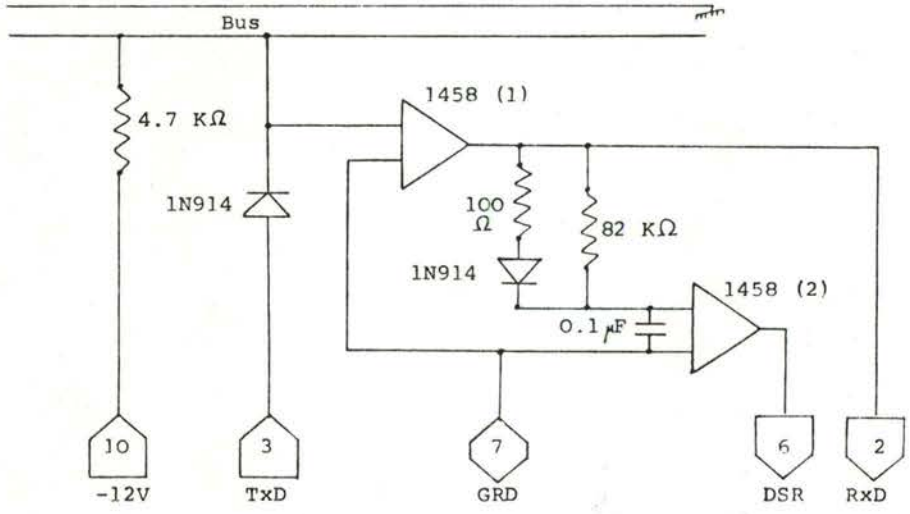


Fig. V.20 L'interface bus.

- Finalement, pour pouvoir déterminer l'état du bus (Signal DSR : 0=occupé, 1=bus libre), on a besoin d'un circuit qui détecte la présence de bytes sur le bus. Ce circuit est constitué d'un monostable à retard qui s'enclenche (Etat instable) au début d'un byte et qui se déclenche (Etat stable) lorsqu'aucun byte ne circule plus sur le bus.
  En fait, le monostable est constitué d'une capacité qui se charge (Enclenchement) lorsqu'une tension positive apparaît sur le bus (Passage d'un "0") ; cette

capacité se décharge petit à petit pour enfin être complètement déchargée : c'est alors son déclenchement.

La transmission en mode asynchrone est telle que chaque byte émis commence précisemment par une tension +12V (Le start-bit = "0"). Quant le bus est au repos (Aucun byte ne circule plus) une tension négative est perpétuellement présente, de sorte que le monostable finit par se déclencher.

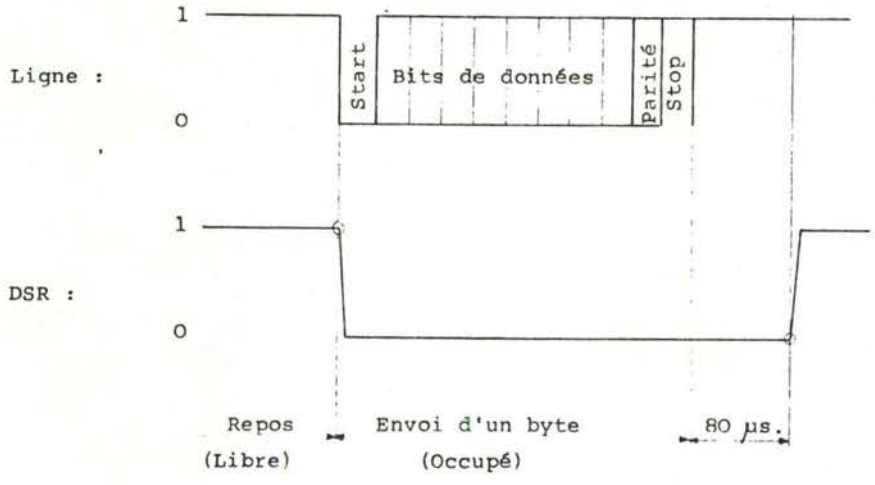


Fig. V.21 Etat de la ligne et du monostable.

En résumé, l'interface-bus réalise l'adaptation entre un ensemble de signaux du type RS-232, et un signal "Bus" (Fig. V.22).

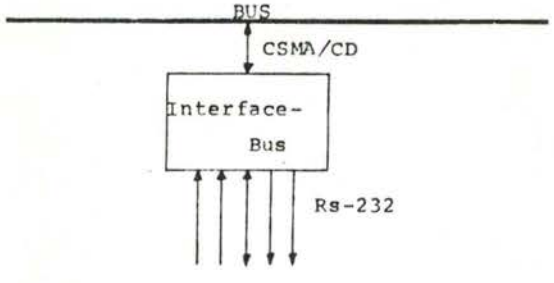


Fig. V.22 Utilité de l'interface-bus.

3.5. Conception software du NIU

Les protocoles propres aux couches transport et liaison de données sont, eux, réalisés sous forme de programmes.
La conception de ces programmes diffère suivant qu'il s'agit de la réception ou de l'émission d'un message.

3.5.1. Logiciel de réception

L'asynchronisme (Voir V.3.2.1.3) des réceptions exige que le PA soit interrompu et qu'un traitement adéquat soit effectué lors de l'arrivée d'un nouveau byte (Réception-prête) ; ce traitement, effectué par un serveur d'interruption de la couche liaison de données, consiste à essayer de reconstituer une trame complète. Lorsque le traitement du byte reçu est terminé, le serveur d'interruption rend immédiatement la main au PA interrompu.

Si le byte reçu permet à la couche liaison de données d'obtenir une trame complète, cette trame est transmise à la couche transport. Cette dernière détermine alors l'acquittement à envoyer et sauve, si nécessaire, le message reçu dans le panier. La couche liaison de donnée se charge enfin de l'envoi de l'acquittement, et le service d'interruption se termine.

En d'autres mots, les traitements qui vont de la réception d'un byte jusqu'au stockage du message dans le panier s'exécutent sous interruptions (Interrupt driven) ; les traitements propres au retrait d'un message du panier (RECEIVE) sont réalisés par des procédures auxquelles les PA font appel.

3.5.2. Logiciel d'émission

Les traitements propres à l'émission de messages sont déclenchés (Synchronisme des réceptions) lors de l'appel du PA à une procédure (SEND).

Une trame est constituée par la couche transport et est fournie ensuite à la couche liaison de données pour envoi.
La couche liaison de données attend d'abord que le bus soit libre avant de commencer l'envoi.

L'envoi de la trame, byte par byte, est effectué par un serveur associé à l'interruption émission-prête.
La détection des collisions est réalisée par un autre serveur, associé à l'interruption réception-prête, qui vérifie la correspondance entre le byte émis et le byte reçu . Si une différence apparaît, le serveur stoppe immédiatement l'émission et la couche transport est avertie de l'avortement du service. Dans le cas contraire, le serveur de détection de collision est automatiquement déconnecté si aucune collision n'est apparue au cours de l'émission des trois premiers bytes.

N.B. :

1. L'enchainement des traitements sous forme d'appel à des procédures Pascal est donné en figure V.23.

2. Les serveurs d'interruptions sont écrits en assembleur (Annexe C2)

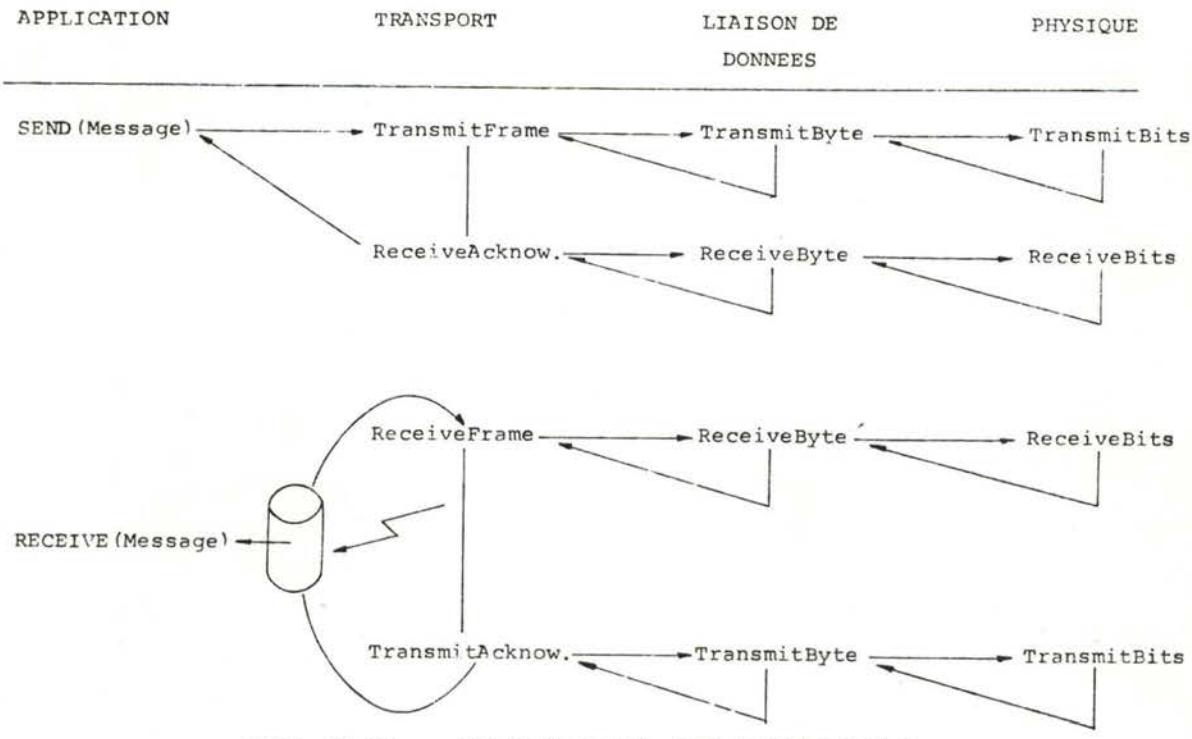


Fig. V.23 Enchainement des traitements.

3.5.3. Principe des échanges

Le principe des échanges peut être explicité sous forme d'un diagramme d'état (Figure V.24 : les commandes effectuées par le PA sont précédées d'un "*", les évènements sont eux précédés d'un "!").

1. L'état inactif : le NIU ne particpe à aucun échange d'information tant que la ligne n'a pas été initialisée (*OPEN_LINK). De même, il ne participe plus aux échanges dès que la ligne a été fermée (*CLOSE_LINK).

2. L'état de repos : dès que la ligne est initialisée, le NIU est en attente soit de l'envoi d'un message (*SEND) soit de la réception d'un nouveau message (!Drapeau).

3. L'état de réception : on passe dans cet état dès la réception d'une nouvelle trame (!Drapeau). On repasse à l'état de repos soit après réception d'une trame complète, soit lorsqu'on a remarqué que la trame n'est pas destinée au NIU.

4. L'état d'envoi : On passe dans cet état lorsque le PA fait une demande d'envoi de message (*SEND). On repasse à l'état

de repos soit lorsque l'envoi a été entièrement réalisé, soit après avortement (Détection de collision).

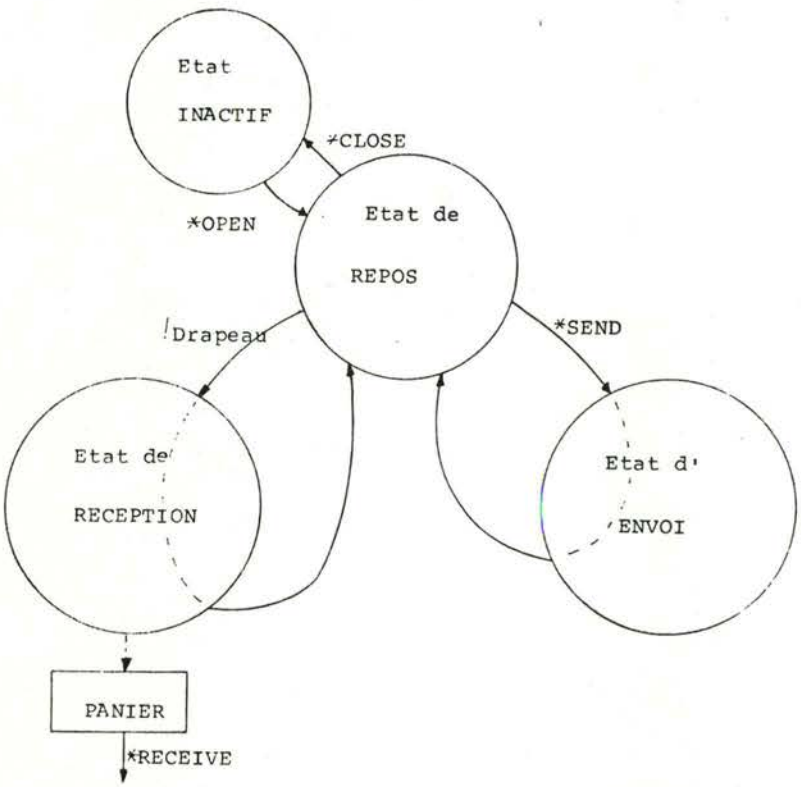


Fig. V.24 Diagrammes des échanges.

CONCLUSION

CONCLUSION

Le but de ce mémoire était d'étudier les possibilités de réalisation d'un réseau local au sein de l'Institut d'Informatique des FNDP.

L'étude approfondie de réseaux locaux existant et d'autres projets devaient aider la réalisation entreprise.

Le projet de stage (Université de Lille), qui consistait en la réalisation d'un logiciel de communication, a permis de se familiariser avec les problèmes qui surviennent dans pareilles réalisations tant du point de vue logique que physique.

La réalisation d'un prototype de réseau local présentait trois aspects essentiels. D'une part, il convenait d'établir une structuration en couches de l'architecture de base de ce réseau. D'autre part, le choix d'une architecture du type bus nécessitait la conception matérielle d'un interface ; ceci constituait un attrait nouveau. Enfin, il a fallu développer un protocole de communication propre à une architecture du type bus ; les travaux de stage ont été d'une grande aide dans ce dernier point.

Au terme de ceci, on peut dresser la situation finale de la réalisation du réseau local.

- Cette réalisation a exigé la conception d'un interface hardware spécialisé (L'interface-bus) ;chacune des fonctions réalisées par cet interface a été testée et vérifiée ;

- Un protocole de communication, qui sied à l'architecture retenue, a été élaboré ;

- Enfin, un logiciel de communication, permettant à des processus d'application d'accéder au réseau, a été réalisé.

Il conviendrait de développer l'étude de ce réseau local dans les directions suivantes :

- Etude des problèmes de fiabilité et estimation des performances du réseau ;

- Augmentation du nombre des fonctions de la couche transport

- Etude des possibilités de connexion d'autres équipements informatiques ; par exemple un serveur disque ou imprimante, un PDP 11 ;

- Etude des besoins d'utilisation d'un réseau local au sein de l'Institut d'Informatique (Par exemple le courrier électronique).

BIBLIOGRAPHIE

Note : Une bibliographie complète en matière de réseaux locaux [SHOCH 1] figure en annexe [Annexe A].

[ANSI] AMERICAN NATIONAL STANDARD INSTITUTE
"Data processing - Open system interconnection - Basic reference model"
ANSI ISO/TC97/SC16
Computer Networks 5 (1981) 81-118

[ANDERSON] ANDERSON G., JENSEN E.
"Computer interconnection structures : taxonomy, characteristics and examples"
Computing survey, december 1975

[BAUWENS] BAUWENS, HENNEBELLE en DE RUYTER
"Ringnetwerk"
PDBCH 281180, november 1980

[BOGGS] BOGGS D., SHOCH J., TAFT E. and METCALFE R.
"PUP : An internetwork architecture"
XEROX CSL 79-10, july 1979

[BOUHOT] BOUHOT J-P., LUSSATO B., FRANCE B.
"La micro-informatique : introduction aux systèmes répartis"
Ed. d'informatique, Paris, 1974

[CII 1] CII - HONEYWELL BULL
"Transmission de données : procédure TMM-RB"

[CII 2] CII _ HONEYWELL BULL
"Procédure de transmission TMM-RB (EBCDIC - ASCII)

[CLARK] CLARK D., POGRAN K., REED D.
"An introduction to local area networks"
Proceedings of the IEEE, vol 66 no. 11, november 1978

[CORDONNIER] CORDONNIER V., RAUCH A.
"L'emploi des mémoires circulantes dans l'architecture d'ordinateurs"
Publication no. 18, Lille - IEEA, juin 1981

[CORNAFION] CORNAFION
"Systèmes informatiques rèpartis : concepts et techniques"
DUNOD, Paris 1981

[COTTON] COTTON I.
"Technologies for local area computer networks"
Proceedings of LACN Symposium, Boston, may 1979, p 25-44

[DANTHINE] DANTHINE A., MAGNEE F.
"Les rèseaux locaux - perspectives et problèmes"
Congrès de la F.A.I.B. "Impact de l'informatique distribuée"
Bruxelles, 7-8 octobre 1980

[DAUGHERTY] DAUGHERTY D., CLEMENT K.
"Ultra-low-cost network for personnal computer"
BYTE Publication, october 1981,p 50-66

[DATTIN] DATTIN X.
"L'assurance de l'informatique"
Informatique et Gestion, no. 126, septembre 1981, p 47-57

[DESJARDINS] DESJARDINS
"Overview ans status of the ISO reference, model of Open System Interconnection"
Computer network 5 (1981) p 77-80

[DIGITAL] DIGITAL, INTEL, XEROX
"The Ethernet : a local area network. Data link layer and physical layer specifications"
Ethernet Specifications version 1.0
XEROX CSL 80-9, september 30, 1980

[ECMA] EUROPEAN COMPUTERS MANUFACTURERS ASSOCIATION
"Networks Layer Principles"
ECMA/TC24/82/18, january 1982

"Local Area Networks Layers 1 to 4 : Architecture and Interconnexion"
ECMA/TC24/82/22, february 1982

"Local Area Networks Link Protocol (CSMA-CD Baseband)
ECMA/TC24/82/24, february 1982

[FARBER] FARBER D., LYLE M., MOKAPETRIS P.
"On the design of local network interfaces"
IFIP Congress Proceedings 1977, p 427-430

[HAFNER] HAFNER E., NENADAL Z., TSCHANZ M.

"A digital loop communication system"
IEEE Transactions on communications, june 1980, p 877-881

[INTEL 1] INTEL CORPORATION
"Ethernet communications controller"
ISBC 550 Hardware reference manual, 1981

[INTEL 2] INTEL CORPORATION
"PLM-80 Programming manual"
"8080/8085 assembly language programming manual"
"RMX/80 user guide"
"ISBC 544 Control board manual"

[JACOBSEN] JACOBSEN T.
"The ISO reference model of Open Systems interconnection"
Networks 80, London, june 1980, p 431-451

[LECOUFFE] LECOUFFE M.P.
"Traitement dynamique de programmes"
Publication no. 13, Lille IEEA, février 1981

[LIU 1] LIU M., OH Y.
"Interface design for Distributed Control Loop Networks (DCLN)"
National Telecommunication Conference, Proceedings 1977, december 1977

[LIU 2] LIU M., REAWES C.
"A loop network for simultaneous transmission of variable length messages"
International conference on parallel processing, IEEE EH 0127-1, september 1977, p 3.31-3.36

[LOBELLE 1] LOBELLE M.
"The TROUT modem"
Technical note no. 19/1, UCL, october 1980

[LOBELLE 2] LOBELLE M.
"The TROUT local network : preliminary general description"
Technical note no. 20/1, UCL, november 1980

[LOBELLE 3] LOBELLE M.
"Comparison of local network architectures"
Technical note no. 21/1, UCL, december 1980

[LORRAINS] LORRAINS
"Réseaux téléinformatiques : les protocoles de transmission"

Hachette technique, 1979, chap 4.5

[LOVELAND] LOVELAND R., STEIN C.
"Fonctionnement du logiciel de réseau DECNET"
Marketing Communication Department, vol IV no. 2, may 1979

[MACCHI] MACCHI G., GUILBERT J-F.
"Transport et traitement de l'information dans les réseaux et systèmes informatiques"
DUNOD, Paris 1979

[MAYOR] MAYOR M.
"A language for network analysis and definition"
ACM sigplan notices, vol 15 no. 1, january 1980, p 130-138

[METCALFE] METCALFE R., BOGGS D.
"Ethernet : distributed packet switching for local computer networks"
Ethernet specifications, Technical report CSL 75-7, may 1975

[MONAHAN] MONAHAN J.
"Opportunities and challenges in the evolution of local area communications networks"
Proceedings of the local area communication network symposium
Boston, may 1979, p 15-24

[MONNIER] MONNIER P.
"Ethernet : réseau local à haute vitesse, ouvert sur l'extérieur"
MINI et MICROS no. 144, p 31-36

[NEEDHAM] NEEDHAM R.
"Systems aspects of the Cambridge ring"
ACM Communications, 12 december 1979, p 82-85

[NICOUD] NICOUD J.D.
"Benefits of a working local network"
IRIA international workshop, "Integrated office system"
Versailles, 6-8 novembre 1979

[NORTH-STAR] NORTH STAR COMPUTER INC.
"System software manual 1979", Soft doc revision 2.1
"Horizon computer system 1978", HRZ-O-DOC revision 1
"North star Z80 processor board ZPB-A 1977", ZPB-DOC rev 3
"North star 16K RAM board 1978", RAM-16-DOC rev 2
"North star 32K RAM board 1979", RAM-32-DOC rev 1

[PENNEY] PENNEY B., BAGHDADI A.
"Survey of computer communications loop networks"
Computer communications, vol 2 no.4 august 1979 p 165-180
vol 2 no.5 october 1979 p 224-242

[PETITPREZ] PETITPREZ
"A flexible circulating memory for communication in a multiprocessor"
Publication no. 12, Lille IEEA, mai 1980

[POUZIN] POUZIN L., ZIMMERMANN H.
"A tutorial on protocols"
Proceedings of the IEEE, vol 66 no. 11, november 1978, p 1346-1370

[SHOCH 1] SHOCH J.
"An annotated bibliography on local computer networks"
Third publication
XEROX, CR categories 3.81

[SHOCH 2] SHOCH J.
"Design and performance of local computers networks"
University microfilms, august 1979

[SOMMER 1] SOMMER R.
"COBUS, a firmware controlled data transmission system"
Second symposium on micro architecture,
Euromicro 1976, p 299-304

[SOMMER 2] SOMMER R.
"A real time protocol for a sub-local network"
International conference on local networks and distributed office systems, London, 11-13 may 1981

[SOMMER 3] SOMMER R.
"A local network interface for LSI 11 computers : LISA "
SWAN Data sheet, EPFL Lausanne, september 1981

[THACKER] THACKER C., Mc CREIGHT E., LAMPSON B., SPROULL R., BOGGS D.
"ALTO : a personnal computer"
XEROX, CR categories 6.21, 8.2
july 11, 1979

[UCSD] UCSD
"PASCAL Revised version I.46"
Institute for information system, La Jolla, California
april 1978

[UNGERMANN] UNGERMANN-BASS, INC.
"NET/ONE : concepts and facilities"
Internal presentation, 1980

[VAN REMORTEL] VAN REMORTEL J.
"Local data networks"
Bell Telephone MFG CY, Antwerp

[WILKES] WILKES M., WHEELER D.
"The Cambridge digital communication ring"
Local area communication networks
Mitre corporation and NBS, Boston, may 1979

[WITTIE] WITTIE L.
"Communication structure for large networks of micro-computers"
IEEE Transactions on computers, vol C-30 no.4, april 1981
p 264-273

[ZAKS] ZAKS R, LESEA A.
"Microprocessor Interfacing Techniques"
SYBEX 1977,1978, Second Edition, Chap VI.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A LA MISE EN OEUVRE D'UN RESEAU LOCAL

(ANNEXES)

Promoteur : Ph. VAN BASTELAER

Mémoire présenté par

Eric LEMAL

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1981-1982

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

CONTRIBUTION A LA MISE EN OEUVRE D'UN RESEAU LOCAL

(ANNEXES)

Promoteur : Ph. VAN BASTELAER

Mémoire présenté par

Eric LEMAL

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Année académique 1981-1982

ANNEXES

Annexe A : "An Annotated Bibliography on Local Computer Networks", Shoch J.

Projet de Stage

Annexe B1 : Procédure en mode de base.

Annexe B2 : Programmes du logiciel de la procédure TMM-RB.

Annexe B3 : Exemples de session.

Mise en Oeuvre d'un Réseau Local

Annexe C1 : Modèle procédural.

Annexe C2 : Programmes du logiciel de réseau et un exemple d'utilisation.

Divers

Annexe D1 : L'alphabet international n° 5 (ASCII).

Annexe D2 : La carte interface ISBC 544.

ANNEXE A

"An Annotated Bibliography on

Local Computer Networks"

An Annotated Bibliography on Local Computer Networks

Third Edition

by John F. Shoch

1.2. References on local computer networks

[Agrawala, *et al.*, 1977]
A. K. Agrawala, R. M. Bryant, and J. Agre, "Analysis of an Ethernet-like protocol", *Computer Networking Symposium*, IEEE Computer Society and NBS, Gaithersburg, December 1977, pp. 104-111.
Models a specific Ethernet-style design using a separate network processor at each host; this processor can buffer packets and generate low-level acks.

[Agrawala, *et al.*, 1978]
A. K. Agrawala, J. R. Agre, and K. D. Gordon, "The slotted ring vs. the token-controlled ring: a comparative evaluation", *IEEE Computer Software and Applications Conference (COMPSAC 1978)*, Chicago, November 1978, pp. 674-679.

[Almes & Lazowska, 1979]
G. T. Almes and E. D. Lazowska, *The behavior of Ethernet-like computer communications networks*, Tech. Report No. 79-05-01, Dept. of Computer Science, U. of Washington, April 1979. Revised version presented at the *7th Symposium on Operating Systems Principles*, Pacific Grove, December 1979, pp. 66-81.

[American Modem, 1979(?)]
American Modem Corp., *Broadband coaxial data modems* (product brochure), American Modem Corp., Bohemia, New York, 1979(?).
Modems and other equipment for CATV systems. Point-to-point links, polling, etc. Provided some of the equipment for GMAD installations.

[Amiot, 1976]
L. W. Amiot, "Front-ending at Argonne National Laboratory", *Proc. of the [1st] Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Laboratory, Berkeley, California, May 1976, pp. 43-56.
Includes a description of the connection between the large Central Computing Facility (CCF) and the Intra-Laboratory Network (ILN).

[Andersen, 1979]
Steven C. Andersen, "A serial data bus control method", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 37-47.
Sperry Univac Shinpads system: bus with centralized control.

[Anderson, 1973]
George A. Anderson, "Interconnecting a distributed processor system for avionics," *Proc. of the 1st Symposium on Computer Architecture*, Gainesville, Florida, December 1973; published as *Computer Architecture News*, 2:4, December 1973, pp. 11-16.
DP/M: proposal included a serial "global bus" system. Phase violation used to transmit a "synch pulse"; that is used to signal round-robin passing of bus control.

[Anderson, 1975a]
D. R. Anderson, "The EPIC-DPS, a distributed network experiment", *IEEE Electronics and Aerospace Systems Convention (Eascon '75)*, Washington, September 1975, paper 121.
A multi-processor system, using a special ring system for processor-to-memory communication (MMDC). Uses a form of "distributed polling" -- like token passing.

[Anderson, 1975b]
J. B. Anderson, "A method for signalling demand in many-user digital loops," *National Telecommunications Conference*, New Orleans, December 1975, pp. 24-1 - 24-3.
To request the loop, each host adds its address to a special field; using a suitable code, the controller must be able to decompose the result.

[Anderson, *et al.*, 1972]
R. R. Anderson, J. F. Hayes, and D. N. Sherman, "Simulated performance of a ring-switched data network", *IEEE Trans. on Comm.*, 20:3, June 1972, pp. 576-591.
GPSS simulation of a Pierce loop.

[Anderson, *et al.*, 1978]
Edward W. Anderson, Edmunde E. Newhall, and Anastasios N. Venetsanopoulos, "A microprocessor-based controller for a loop switching system", *International Conference on Communications (ICC '78)*, Toronto, June 1978.
Loop with a supervisor provides master clock, initialization of 'control' and recovery. Uses modified HDLC: control sequence is a 0 – seven 1's, changed to an HDLC flag when control is seized. Uses separate loop interface, with buffers etc. – microprocessor + 66 IC's. Only a prototype controller; runs at up to 19.2 kbps.

[Ashenhurst, 1975]
Robert L. Ashenhurst, "Centralized or decentralized computing -- or maybe some of both?", *IIth IEEE Computer Society International Conference (COMPCON Fall '75)*, Washington, September 1975, pp. 55-60.

[Avizienis & Vonderohe, 1975]
A. Avizienis and R. H. Vonderohe, "A hierarchical network", *Datamation*, 21:2, February 1975, pp. 43-44.
Simple hierarchy: local minicomputers tied into an intermediate level (MOM), and ultimately a large host (DAD).

[Avi-Itzhak, 1971]
B. Avi-Itzhak, "Heavy traffic characteristics of a circular data network", *Bell System Technical Journal*, 50:6, October 1971, pp. 2521-2549.
Has a forward reference to [Pierce, 1972b], which later outlined the design of this "empty slot" ring.

[Babic, *et al.*, 1977]
Gojko A. Babic, Ming T. Liu, and Roberto Pardo, "A performance study of the distributed loop computer network (DLCN)", *Computer Networking Symposium*, IEEE Computer Society and NBS, Gaithersburg, December 1977, pp. 66-76.

[Bain, 1978]
Lee L. Bain, "N-way architecture -- clustered Sigma/CP-V mainframes share common file and peripheral resources," *Trends and Applications: 1978, Distributed Processing*, IEEE(CS)/NBS, Gaithersburg, May 1978.
Specialized switch providing access from multiple hosts to a set of peripherals.

[Barber, 1973]
D. L. A. Barber, "Local data networks", in Grimsdale and Kuo, eds., *Computer Communication Networks (Proc. of the NATO Advanced Study Institute on Computer Communication Networks, Sussex, September 1973)*, Noordhoff, 1975.
Very general introduction, and a bit of detail on the NPL local network.

[Barkauskas, *et al.*, 1973]
B. J. Barkauskas, R. R. Renac, and C. A. Trlica, "A computer network for peripheral time sharing", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, February 1973, pp. 227-229.
BTL-Naperville, star configuration, mini-computers given access to peripherals on the central machine.

[Bartlett, 1968]
K. A. Bartlett, "Transmission control in a local data network", *IFIP Congress 68*, Edinburgh, August 1968.
More on the use of multiplexers to reach the single switch in a local area of the NPL proposal.

[Beaston, 1978]
John Beaston, "LSI devices control loop-mode SDLC data links", *Data Communications*, August 1978, pp. 65-72.

[Biba & Yeh, 1979]
K. J. Biba and J. W. Yeh, "Fordnet: A front-end approach to local computer networks", *Local Area Communicctions Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 199-215.
Ford Aerospace, Palo Alto. Uses PDP-11's with a separate controller/interface (the UMC-Z80 which includes a Zilog Z80 and SIO chip); runs at 880 kbps, and connects to the Ford Motor Co. transceiver.

[Bilek, *et al.*, 1978]
R. W. Bilek, D. A. Lutzky, and J. J. Peterka, "Simulating a local computer network", *Third Conference on Local Computer Networking*, U. of Minn., Minneapolis, October 1978.
Discrete event simulation of a Hyperchannel system; very limited results, but did show potential deadlocks in allocation of channel adapters.

[Blauman, 1979]
S. Blauman, "Labeled slot multiplexing: a technique for a high speed, fiber optic based, loop network," *Proc. of the 4th Berkeley Conference on Distributed Data Management and Computer Networks*, San Francisco, August 1979, pp. 309-321.
A fiber optic ring, with no central control, 20 Mbps. Very small packets, however, which are fully buffered in every interface around the ring.

[Bliss, *et al.*, 1976]
B. B. Bliss, W. A. Counterman, and E. A. Mackey, "Proposal for a ring network -- IDANET", *Conference on Experiments in New Approaches to Local Computer Networking*, U. of Minn., St. Paul, September 1976.
A very similar version of this paper was also presented at the same conference the following year.

[Bock, 1977]
Peter Bock, "A data communications operating system (DCOS) for microprocessor-driven peripherals", *16th Annual Technical Symposium*, ACM(DC) and NBS, Gaithersburg, Maryland, June 1977, pp. 159-166.
Simple star to share peripherals among a small number of hosts; uses Altair as a switch, operator control to set up connections, runs at up to 9600 kbps.

[Boggs & Metcalfe, 1978]
David R. Boggs and Robert M. Metcalfe, *Communications Network Repeater*, US Patent no. 4,099,024, July 1978.

[Brandenburg, *et al.*, 1972]
L. H. Brandeburg, B. Gopinath, and R. P. Kurshan, "On the addressing problem of loop switching", *Bell System Technical Journal*, 50:7, September 1972, pp. 1445-1469.
More on addressing methods for inter-loop switching. Uses matrix algebra techniques to analyse the network; provides a more space-efficient method.

[Brandenburg & Gopinath, 1972]
L. H. Brandenburg and B. Gopinath, "A table look up approach to loop switching", *Bell System Technical Journal*, 51:9, November 1972, pp. 2095-2099.
More on inter-loop switching, this time using a scaler product of the address in the block, with a bit entry in a special table. Yields the distance to the destination.

[Brickner, 1979]
D. R. Brickner, "Military multiplex standard defines versatile serial bus," *Computer Design*, December 1979, pp. 93-99.
The 1553B military standard: serial bus supporting up to 31 devices, with a designated bus controller. Applications include aircraft, helicopters, and the space shuttle.

[Brinton, 1980]
J. B. Brinton, "Data network switches packets," *Electronics*, February 14, 1980, pp. 183-184.
DATANET phase III: finally introduced routing through intermediate hosts.

[Burnett & Sethi, 1977]
D. J. Burnett and H. R. Sethi, "Packet switching at Philips Research Laboratories", *Computer Networks*, 1977, pp. 341-348.
Hosts all linked to a central packet switch.

[Bylander, 1979]
J. R. Bylander, "Integrated services using coaxial cable," *Proc. of the National Electronics Conference*, Chicago, October 1979, pp. 495-497.
3M's CS² system: single-cable CATV; major emphasis on carrying voice and video (esp. in rural areas), but can also carry data. Installed at the 3M HQ.

[Cain & Morling, 1978]
G. D. Cain and R. C. S. Morling, "Mininet: a local area network for real-time instrumentation and control", *Third Conference on Local Computer Networks*, U. of Minn., Minneapolis, October 1978.
A special-purpose system for process control devices. Very small packets, virtual circuits, store-and-forward via microprocessor "exchanges."

[Carpenter & Rosenthal, 1978]
Robert J. Carpenter and Robert Rosenthal, "A local network for the National Bureau of Standards", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 7-9.

[Carpenter & Sokol, 1979]
R. J. Carpenter and J. Sokol, Jr., "Serving users with a local area network", *Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 75-85.

[Carpenter, *et al.*, 1978]
Robert J. Carpenter, Joseph Sokol, Jr., and Robert Rosenthal, "A microprocessor-based local network node", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, September 1978, pp. 104-109.
Microprocessor TIEs for dumb devices; packet buffer in the TIE, user interface runs only up to 9600 bps.

[Carsten & Posner, 1978]
R. T. Carsten and M. J. M. Posner, "Simplified statistical models of single and multiple Newhall loops", *National Telecommunications Conference (NTC '78)*, Birmingham, December 1978, pp. 44.5.1-44.5.7.

[Carsten, *et al.*, 1977]
R. T. Carsten, E. E. Newhall, and M. J. M. Posner, "A simplified analysis of scan times in an asymmetrical Newhall loop with exhaustive service", *IEEE Trans. on Comm.*, com-25:9, September 1977, pp. 951-957.

[Chang & Roberts, 1979]
E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes", *CACM*, 22:5, May 1979, pp. 281-283.
Describes an algorithm that might be used to assure unique regeneration of a lost control token, in a token-passing ring system.

[Chesson, 1979]
G. L. Chesson, "Datakit software architecture", *International Conference on Communications (ICC '79)*, Boston, June 1979, paper 20.2.
See also [Fraser, 1979].

[Chesson & Fraser, 1980]
G. L. Chesson and A. G. Fraser, "Datakit network architecture", *20th IEEE Computer Soc. Int. Conf. (Compcon Spring '80)*, San Francisco, February, 1980, pp. 59-61.
Further discussion of Datakit; indicates that interconnected Datakit systems must synchronize to a master clock.

[Christensen, 1977]
Gary Christensen, "Data trunk contention in the Hyperchannel network", *Conference on "A Second Look at Local Computer Networking"*, U. of Minn., Minneapolis, October 1977, p. 10.

[Christensen, 1978a]
Gary S. Christensen, "Applications of Hyperchannel", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978.

[Christensen, 1978b]
Gary S. Christensen, "Network monitor unit", *Third Conference on Local Computer Networks*, U. of Minn., Minneapolis, October 1978.

[Christensen, 1979a]
Gary S. Christensen, "Links between computer-room networks", *Telecommunications*, February 1979, pp. 47-50.
A summary of the Hyperchannel and its adapters; description of the "link adapters" which connect to T-carrier channels, thus linking 2 Hyperchannel systems.

[Christensen, 1979b]
Gary S. Christensen, "A network storage system", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 86-90.
Proposal for a network file server, using the NSC Hyperchannel.

[Christensen & Franta, 1978]
Gary S. Christensen and W. R. Franta, "Design and analysis of the access protocol for Hyperchannel networks", *3rd USA-Japan Computer Conference*, San Francisco, October 1978, pp. 86-93.
This paper is actually a consolidation of two different papers which were presented at an earlier conference in Minneapolis. The first part is an overview of the Hyperchannel, which is not bad; the second part is a bit of analysis, and is not as strong.

[Christman, 1973]
Ronald D. Christman, "Development of the LASL computer network", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, February 1973, pp. 239-242.
A star configuration, to service terminal users and large machines; controlled from a single Front End Machine (FREM).

[Chu, 1974]
W. W. Chu, ed., *Advances in Computer Communications*, Artech House, Dedham, Mass., 1974.

[Chu, 1976]
W. W. Chu, ed., *Advances in Computer Communications, 2nd edition*, Artech House, Dedham, Mass., 1976.
Contains all of the articles from [Chu, 1974], plus some new ones. Both editions include reprints of [Abramson, 1973b; Roberts, 1973; Yuen, *et al.*, 1972; Farber, *et al.*, 1973; Pierce, 1972a; Hayes & Sherman, 1971b; Davies, 1966b; Scantlebury & Wilkinson, 1971; Heart, *et al.*, 1970].

[Chu, 1979]
W. W. Chu, ed., *Advances in Computer Communications and Networking, Revised 3rd edition*, Artech House, Dedham, Mass., 1979.
Includes reprings of [Abramson, 1977; Kleinrock & Gerla, 1978; Kahn, 1975; Clark, *et al.*, 1978; Metcalfe & Boggs, 1976].

[Chu & Konheim, 1972]
W. W. Chu and A. G. Konheim, "On the analysis and modeling of a class of computer communications systems", *IEEE Trans. on Comm.*, com-20:3, June 1972, pp. 645-660.
Includes a discussion of analytical results for loops.

[Clark, *et al.*, 1978]
David D. Clark, Kenneth T. Pogran, and David P. Reed, "An introduction to local area networks", *Proc. of the IEEE*, 66:11, November 1978, pp. 1497-1517.

[Coker, 1972]
C. H. Coker, "An experimental interconnection of computers through a loop transmission system", *Bell System Technical Journal*, 51:6, July-August 1972, pp. 1167-1175.
Written October 1971; describes the interface to the two host computers. Honeywell DDP516, Bell Labs Acoustic Research Facility, 16k, 16bit, ~1 microsec. memory. Did an FTP program, to get files from the other machine; used a Positive Retransmission scheme. Also did remote loading and running of the second machine. Max. user data rate: 56 Kbits/sec.

[Computrol, 1979(?)]
Computrol, *High speed digital communication products*, product brochure, Computrol, Ridgefield, Conn., 1979(?).
A 1 Mbps interface for LSI-11s; uses a shared coax, with SDLC implemented in the hardware. Incorporates their own FSK modem.

[Conant & Wecker, 1976]
G. Conant and S. Wecker, "DNA: an architecture for heterogeneous computer networks", *Third International Conference on Computer Communication (ICCC)*, Toronto, August 1976, pp. 618-625.

[Cotton, 1979]
I. W. Cotton, "Technologies for local area computer networks," *Proc. of the Local Area Communications Network Symposium*, Boston, May 1979, pp. 25-45.

[Crane & Taft, 1980]
R. C. Crane and E. A. Taft, "Practical considerations in Ethernet local network design," *Proc. of the 13th Hawaii International Conference on Systems Sciences*, Honolulu, January 1980, pp. 166-174.

[Crowther, *et al.*, 1975]
W. R. Crowther, F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, "Issues in packet switching network design", *AFIPS Conference Proceedings -- 1975 NCC*, 44, Anaheim, May 1975, pp. 161-175.
An expanded version of this paper later appeared as [McQuillan & Walden, 1977].

[Cullum, 1976]
P. G. Cullum, "The transmission subsystem in Systems Network Architecture", *IBM Systems Journal*, 15:1, 1976, pp. 24-38.

[Cypser, 1978]
R. J. Cypser, *Communications architecture for distributed systems*, Addison-Wesley, 1978.
Presents a detailed discussion of IBM's SNA.

[Data General, 1977]
Data General Corp., *4206 Series Multiprocessor Communications Adapter*, 014-000072-01, Rev. 01, July 1977.
A daisy-chained bus for up to 15 DG hosts; 16-bit parallel data, 47 wires in all.

[Davies, *et al.*, 1967]
D. W. Davies, K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A digital communication network for computers giving rapid response at remote terminals", *ACM Symposium on Operating System Principles*, Gatlinburg, Tenessee, October 1967.
The original paper on the NPL proposal.

[Davies, 1968a]
D. W. Davies, "Communication networks to serve rapid-response computers", *IFIP Congress 68*, Edinburgh, August 1968.

[Davies, 1968b]
D. W. Davies, "The principles of a data communication network for computers and remote peripherals", *IFIP Congress 68*, Edinburgh, August 1968. Reprinted in [Chu, 1976].
Describes the two-tier system, with a packet-switched backbone and a single switch in each local area.

[Davies, 1971]
D. W. Davies, "Packet switching in a public data network", *Information Processing 71 (Proc. of IFIP 71)*, Ljubljana, August 1971.
Mentions the single-switch configuration at NPL.

[Davies & Barber, 1973]
Donald W. Davies and Derek L. A. Barber, *Communication Networks for Computers*, John Wiley & Sons, 1973.
Esp. pp. 261-267, on the NPL local network: a tree of multiplexers with a single switch.

[Davis, 1979]
George R. Davis, "The changing face of the private branch exchange," *Data Communications*, August 1979, pp. 43-49.
A general review of newer PBXs using digital switching, and their potential to serve as star-shaped switches handling digital data connections.

[Davison, 1978]
Alan Davision, *Design of a low-cost broadcast packet transmission network*, TR 119, Computer Systems Laboratory, Queen Mary College, October 1977, revised March 1978.

[DEC, 1978a]
Digital Equipment Corporation, *CSS Products: PCL11-B parallel communications link*, Product Bulletin 10.4, March 1978.
The "parallel communications link" for the PDP-11; uses TDM control of a 16-bit bus, supporting up to 16 PDP-11s. Maximum length of 300 feet.

[DEC, 1978b]
Digital Equipment Corporation, *Decnet phase II, networking distributed computers, a progress report, March 1978*, 1978.

[DeMarines & Hill, 1976]
Victor A. DeMarines and Lawrence W. Hill, "The cable bus in data communications", *Datamation*, August 1976.

[DeMarines & Willard, 1978]
Victor A. DeMarines and David Willard, "Use of CATV coaxial cable supported bus structures", *ACM 1978 Annual Conference*, Washington, DC, December 1978, pp. 478-479.

[Dickey, 1974]
Shane Dickey, "Distributed computer systems", *Hewlett-Packard Journal*, November 1974, pp. 2-11.
Star shaped system; precursor to later HP DS/1000 work.

[Dickinson, 1979]
R. V. C. Dickinson, "A versatile, low cost system for implementing CATV auxiliary services," *National Cable Television Association Convention*, Las Vegas, 1979, pp. 65-72.
A simple polling system, with central control. From E-Com Corp.

[Diffley, 1973]
Michael W. Diffley, "Design considerations of a proposed local area computer network emphasizing the needs of the health sciences", *3rd Data Communications Symposium*, ACM and IEEE, St. Petersburg, Florida, November 1973, pp. 97-103.
Currently running a very small star; proposal for a small Arpanet-style system, with "exchange nodes".

[Dineson & Picazo, 1980]
M. A. Dineson and J. J. Picazo, "Broadband technology magnifies local networking capability," *Data Communications*, February 1980, pp. 61-79.
Discusses use of CATV systems for local networking; generally uses a modulated RF carrier.

[Dixon, *et al.*, 1972]
R. C. Dixon, L. J. Hash, J. D. Markov, and L. P. West, "Process to release and reuse time slots in a loop communications system," *IBM Technical Disclosure Bulletin*, 15:1, June 1972, pp. 335-336.
Centralized terminal system with an "empty slot" approach; 1-address messages to or from a control unit.

[Donnan & Kersey, 1974]
R. A. Donnan and J. R. Kersey, "Synchronous data link control: A perspective", *IBM Systems Journal*, 13:2, 1974. Reprinted in [Green & Lucky, 1975].

[Donnelley & Yeh, 1978a]
James E. Donnelley and Jeffry W. Yeh, "Interaction between protocol levels in a prioritized CSMA broadcast network", *Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Laboratory, San Francisco, August 1978. Reprinted in *Computer Networks*, 3, 1979, pp. 9-23.

[Donnelley & Yeh, 1978b]
James E. Donnelley and Jeffry W. Yeh, "Simulation studies of round robin contention in a prioritized CSMA broadcast network", *Third Conference on Local Computer Networking*, U. of Minn., Minneapolis, October 1978.
Further simulations of proposed modifications to the Hyperchannel mechanisms; done at LLL.

[Dragoo, 1979]
R. E. Dragoo, "Fiber optic data bus," *National Electronics Conference*, Chicago, October 1979, pp. 179-184.
Design study for a fiber optic 30-port transmissive star coupler, to run at 10 Mbps; very similar to the Fibernet design. Intended for aircraft applications similar to MIL-STD-1553B.

[Elenbaas & King, 1977]
A. E. Elenbaas and L. L. King, "High performance local communications based on the CCITT X.25 protocol," *Proc. of the Computer Networking Symposium*, IEEE (CS)/NBS, Gaithersburg, December 1977, pp. 76-82.
Proposal for a store-and-forward system using separate switching nodes, based upon the X.25 protocol.

[Farber, 1970]
D. J. Farber, "Data ring oriented computer networks", in Rustin, ed., *Computer Networks (Courant Computer Science Symposium 3, November-December 1970)*, Prentice-Hall, New York, 1972, pp 79-94.
A fascinating early paper: no central control, but fixed size blocks on the loop (300 bits); each with a leading free/busy bit. Direct, 8-bit addressing of destination node.

AN ANNOTATED BIBLIOGRAPHY ON LOCAL COMPUTER NETWORKS

[Farber, 1972]
D. J. Farber, "Networks, an introduction", *Datamation*, April 1972, pp. 36-39. Reprinted in [Green & Lucky, 1975].
A comparison of 7 networks, including DCS. Written in the "future present tense," it reports 9 nodes on the ring, using fixed length messages; that configuration was apparently never operational.

[Farber, 1974]
D. J. Farber, "An overview of distributed processing aims", *9th Annual [sic] IEEE Computer Society International Conference (COMPCON Fall 74)*, Washington, September 1974, pp. 191-193.

[Farber, 1975a]
D. J. Farber, "A ring network", *Datamation*, February 1975, pp. 44-46.

[Farber, 1975b]
D. J. Farber, "A distributed computer system -- an overview", *Proceedings of the National Electronics Conference*, 30, Chicago, Oct. 1975, pp. 188-190.

[Farber, 1977]
David J. Farber, "The ARPA local network interface", *IEEE Electronics and Aerospace Systems Convention (Eascon '77)*, Arlington, Va., September 1977, paper 14-3.

[Farber & Heinrich, 1972]
D. J. Farber, and F. R. Heinrich, "The structure of a distributed computer system -- the distributed file system", *First International Conference on Computer Communication (ICCC)*, Washington, October 1972, pp. 364-370.
Elaborates upon the file system, as part of DCS.

[Farber & Larson, 1972a]
David J. Farber and Kenneth C. Larson, "The system architecture of the distributed computer system -- the communication system", *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 21-27.
Fixed length, empty slot approach.

[Farber & Larson, 1972b]
David J. Farber and Kenneth C. Larson, "The structure of a distributed computing system -- software", *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 539-545.

[Farber & Vittal, 1973]
D. J. Farber, and J. J. Vittal, "Extendability considerations in the design of the Distributed Computer System (DCS)", *National Telecommunications Conference (NTC 73)*, Atlanta, November 1973, pp. 15E.1-15E.6.

[Farber, *et al.*, 1973]
D. J. Farber, J. Feldman, F. R. Heinrich, M. D. Hopwood, K. C. Larson, D. C. Loomis, and L. A. Rowe, "The Distributed Computing System", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, February 1973, pp. 31-34. Reprinted in [Chu, 1976].

[Farmer & Newhall, 1969]
W. D. Farmer and E. E. Newhall, "An experimental distributed switching system to handle bursty computer traffic", *Proceedings of the ACM Symposium on Problems in the Optimization of Data Communication Systems [1st Data Communications Symposium]*, Pine Mountain, Georgia, October 1969, pp. 1-33.
Describes 3-station prototype of a ring: no central control, but a "loop supervisor" to provide clocking. 6.312 Mhz, 3.156 Mbps, but does not use standard T2 coding: 1 bit delay per host. "Primary" part of interface is powered from the line. Loop supervisor puts 0's on the loop, reclocks signals.

[Farrell, *et al.*, 1979]
E. P. Farrell, N. Ghani, and P. C. Treleaven, "A concurrent computer architecture and a ring based implementation," *6th Annual Symposium on Computer Architecture*, Philadelphia, April 1979, pp. 1-11.
General proposal for a multi-processor, with a ring communications system.

[Figueroa, 1978]
John W. Figueroa, "PHI, the HP-IB interface chip", *HP Journal*, 29:11, July 1978, pp. 16-17.
Describes chip implementation of IEEE 488 standard.

[Fitzwilliam & Wagner, 1978]
J. W. Fitzwilliam and R. L. Wagner, "Transaction network, telephones and terminals: overview", *Bell System Technical Journal*, 57:10, December 1970, pp. 3325-3329.
Transaction system with a central host, via switched network or using a local message switch for polled access.

[Fletcher, 1973a]
John G. Fletcher, "Octopus communications structure", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, San Francisco, February 1973, pp. 21-23.

[Fletcher, 1973b]
J. G. Fletcher, "The Octopus computer network", *Datamation*, April 1973, pp. 58-63.

[Fletcher, 1975]
John G. Fletcher, "Principles of design in the Octopus computer network", *ACM 1975 Annual Convention (ACM '75)*, Minneapolis, October 1975, pp. 325-328.

[Fortune, *et al.*, 1977]
P. J. Fortune, W. P. Lidinsky, and B. R. Zelle, "Design and implementation of a local computer network", *International Conference on Communication (ICC '77)*, Chicago, June 1977, pp. 221-226.
Design for an Arpanet-like system; first phase is a two-host prototype.

[Franck & Patton, 1979]
A. Franck and P. C. Patton, "Some architectural and system implications of local computer networks," *15th IEEE Comp. Soc. Int. Conf. (Compcon '79 Spring)*, San Francisco, February 1979, pp. 272-276D.
Paper mainly discusses the NSC Hyperchannel.

[Franta, 1976]
W. R. Franta, "Early remarks on trunk utilization and message throughput", *Conference on Experiments in New Approaches to Local Computer Networking*, U. of Minn., St. Paul, September 1976.

[Franta, 1977]
W. R. Franta, "Decision and realization points for random access path controls", *Conference on "A Second Look at Local Computer Networking"*, U. of Minn., Minneapolis, October 1977.

[Fraser, 1974a]
A. G. Fraser, "Spider -- an experimental data communications system", *International Conference on Communications (ICC '74)*, Minneapolis, June 1974, pp. 21F-1 - 21F-10.
Buffered, centrally controlled. Central switching machine, connected to terminals with a T1 twisted-pair line (1.544 megabits/sec). Fixed bits around the loop. Each TIU introduces 8 bits of delay.

[Fraser, 1974b]
A. G. Fraser, *Spider -- A data communications experiment*, Bell Laboratories Computing Science Technical Report #23, December 1974.

[Fraser, 1974c]
A. G. Fraser, *Loops for data communications*, Bell Laboratories Computing Science Technical Report #24, December 1974.

[Fraser, 1974d]
A. G. Fraser, "Loop transmission systems for data", *Computer Communications Review*, 4:4, October 1974, pp. 2-8.

[Fraser, 1975]
A. G. Fraser, "A virtual channel network", *Datamation*, 21:2, February 1975, pp. 51-56.

[Fraser, 1979]
A. G. Fraser, "Datakit -- A modular network for synchronous and asynchronous traffic", *International Conference on Communications (ICC '79)*, Boston, June 1979, paper 20.1.
See also [Chesson, 1979]. Up to 511 modules connected in a star to a single node.

[Freeman & Thurber, 1979]
H. A. Freeman and K. J. Thurber, "Issues in local computer networks", *International Conference on Communications (ICC '79)*, Boston, June 1979, paper 20.3.
Revised version of [Thurber & Freeman, 1979a].

[Frisch, 1977]
Ivan T. Frisch, "Experiments on random access packet data transmission on coaxial cable video transmission systems", *IEEE Trans. on Comm.*, com-25:10, October 1977, pp. 1199-1203.
Describes results of experimental transmission of packets through a CATV system.

[Gable, 1978]
M. G. Gable, "A local network architecture for industrial applications", *Instrumentation Society of America 1978 Annual Conference*, pp. 119-124.
Ford Research's bus system for process control, testing, etc.

[Gall & Mueller, 1972]
D. A. Gall and H. R. Mueller, "Waiting-time distributions and buffer overflow in priority queueing systems", *IEEE Trans. on Comm.*, com-20:5, October 1972, pp. 865-877.
Single server loop, with priority polling scan.

[Gerard, 1979]
J. M. Gerard, *The development and use of a high speed packet switching network in a high-energy physics laboratory*, CERN Data Handling Division, Report DD/79/2, April 1979.
A store-and-forward system using separate IMPs. Became operational in 1978.

[Gerard, 1980]
J. M. Gerard, *The effect of the CERNET network on computing at CERN*, CERN Data Handling Division, Report DD/80/5, February 1980.
Store-and-forward system, with 12 packet switches and supporting 40 user machines.

[Gerhardstein, *et al.*, 1978]
L. H. Gerhardstein, J. O. Schroeder, and A. J. Boland, "The Pacific Northwest Laboratory minicomputer network", *Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Laboratory, San Francisco, August 1978.

[Gfeller, *et al.*, 1978]
F. R. Gfeller, H. R. Muller and P. Vettiger, "Infrared communication for in-house applications", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, September 1978, pp. 132-138.
IBM Zurich; using LED's and photodiodes on terminals within a room, with a 'mailbox' on the ceiling.

[Gfeller & Bapst, 1979]
F. R. Gfeller and U. Bapst, "Wireless in-house data communication via diffuse infrared radiation", *Proc. of the IEEE*, 67:11, November 1979, pp. 1474-1486.
More detailed discussion of the work at IBM Zurich: using LED's and photodiodes on terminals within a room, with a 'satellite' on the ceiling. Uses a broadcast channel out, multi-access channel in. Built a limited prototype.

[Ghezzi, *et al.*, 1973]
C. Ghezzi, G. Le Moli, and L. Mezzalira, "Introduction to Poli computer network design," *International Computing Symposium*, 1973, pp. 271-278.
A star, with a "nodal center" which looks like a terminal to each host.

[Gordon & Nelson, 1979]
R. L. Gordon and D. L. Nelson, *The use of rings in high availability local networks*, Prime Computer Inc., Research Department Technical Report, March 2, 1979.

[Gordon, *et al.*, 1979]
R. L. Gordon, W. W. Farr, and P. Levine, "Ringnet: A packet switched local network with decentralized control", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 13-19.
A token passing ring, at 8 Mbps; also called the Primenet Node Controller Ring (PNC).

[Graham & Pollak, 1971]
R. L. Graham and H. O. Pollak, "On the addressing problem for loop switching", *Bell System Technical Journal*, 50:8, October 1971, pp. 2495-2519.
Contains a forward reference to Pierce's then unpublished paper, which did not emerge until the following year. Advocates distributed control and not a pre-determined routing. Suggests special binary addresses for each node; can then compute a Hamming distance between two nodes (loops).

[Grandy & Sargent, 1979]
R. C. Grandy and C. E. Sargent, Jr., "TDM solutions for internal aircraft communications: C^3 concept for E-4 evolution," *Local Area Communication Networks Symposium*, Boston, May 1979, pp. 251-261.
Proposal for an aircraft bus system, to carry data and voice.

[Graube, 1980]
M. Graube, "PROWAY -- A local network for process control," *20th IEEE Comp. Soc. Int. Conf. (Compcon Spring '80)*, San Francisco, February 1980, pp. 313-316.
Describes functional requirements for a process control network. Suggests use of a shared, passive cable; various alternative control schemes.

[Gray, 1977]
J. P. Gray, "Network services in Systems Network Architecture", *IEEE Trans. on Comm.*, com-25:1, January 1977, pp. 104-116.

[Gray & Blair, 1975]
J. P. Gray and C. R. Blair, "IBM's Systems Network Architecture", *Datamation*, April 1975, pp. 51-56.

[Green & Lucky, 1975]
P. E. Green, and R. W. Lucky, eds., *Computer Communications*, IEEE Press, New York, 1975.
Includes reprints of [Doll'an & Kersey, 1974; Switzer, 1972; Abramson, 1973b, Farber, 1972; Pierce, 1972a; West, 1972a; Heart, *et al.*, 1970].

[Greenblatt, 1979]
Richard Greenblatt, personal communication, MIT AI Laboratory, May 1979.
Demonstration of the Chaosnet. There are no papers yet published.

[Hafner, 1974a]
E. R. Hafner, "Digital communication loops - a survey", *International Zurich Seminar on Digital Communication*, March 1974, paper D1.
A good introductory survey; additional details on their "loop extension" approach.

[Hafner, 1974b]
E. R. Hafner, "Implementation of distributed control in a loop system", *International Switching Symposium*, Munchen, 1974, pp. 148/1 - 148/4.
Describes the switching procedures used with the ring to support voice communication: establish calls, carry voice, build conference calls, etc.

[Hafner & Nenadal, 1976]
E. Hafner and Z. Nenadal, "Enhancing the availability of a loop system by meshing", *International Zurich Seminar on Digital Communication*, March 1976, paper D4.
Running alternate lines in a loop structure, able to skip over failed nodes.

[Hafner, *et al.*, 1973]
E. R. Hafner, Z. Nenadal, M. Tschanz, "A digital loop communication system", *International Conference on Communications (ICC '73)*, Seattle, June 1973, pp. 50-24 - 50-29. Revised version published in *IEEE Trans. on Comm.*, June 1974, pp. 877-881.
Switching a shift register into a loop. Distributed control, but still has a special node to provide clock and synch, remove smashed packets. Designed mainly for voice.

[Hanks, 1978]
James P. Hanks, "Mitrenet -- introduction and overview", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, p. 10.

[Hare, 1972]
A. G. Hare, "An integrated wideband communication system for local distribution," *1972 International Zurich Seminar on Integrated Systems for Speech, Video, and Data Communications*, Zurich, March 1972, paper C4.
General discussion of integrating voice, data, video, etc. on a loop network.

[Hare & Ithell, 1975]
A. G. Hare and A. H. Ithell, "Multipurpose wide-band local distribution -- proposals for an integrated system," *IEEE Trans. on Comm.*, com-23:1, January 1975, pp. 42-48.
Cable ring to provide voice, data, video, and other services. Uses FDM and TDM; demonstrated a small prototype.

[Hassing, *et al.*, 1973]
Thomas E. Hassing, Raymond M. Hampton, Gerald W. Bailey, and Robert S. Gardella, "A loop network for general purpose data communications in a heterogeneous world", *3rd Data Communications Symposium*, ACM and IEEE, St. Petersburg, Florida, November 1973.
Done at NSA, two rings running in opposite directions, empty slot technique. Set switches to make one node generate the clock for all, and introduce a null packet, if needed. Packets may circulate many times if not taken immediately at the destination.

[Hatada, et al., 1979]
M. Hatada, K. Hiyama, and H. Ihara, "A microprocessor-based multi-loop network system," *19th IEEE Comp. Soc. Int. Conf. (Compcon Fall '79)*, Washington, DC, September 1979, pp. 454-461.
Ring with a distributed algorithm to allocate access. To get on, hosts circulate their priority number; if it comes back, they have the highest priority, and can use the ring.

[Hayes, 1973]
J. F. Hayes, "Modeling an experimental computer communications network", *3rd Data Communications Symposium*, ACM and IEEE, St. Petersburg, Florida, November 1973, pp. 4-11.
Models what became Spider; similar to [Hayes, 1974].

[Hayes, 1974]
J. F. Hayes, "Performance models of an experimental computer communications network", *Bell System Technical Journal*, 53:2, February 1974, pp. 225-259.
Models what became Spider: unlike Pierce's earlier work, it is a loop to a central switch. Switch does routing and control of traffic. Switch can buffer blocks, and can centrally tell terminals to shut off, if backlog is growing. All data must first go through the switch, even if it is destined for another terminal on the same loop. Uses 1.544 megabit/sec line (T1).

[Hayes & Sherman, 1971a]
J. F. Hayes and D. N. Sherman, "Traffic and delay in a circular data network", *2nd Symposium on Problems in the Optimization of Data Communication Systems [2nd Data Communications Symposium]*, ACM and IEEE, Palo Alto, California, October 1971.
Delay estimates for a Pierce loop.

[Hayes & Sherman, 1971b]
J. F. Hayes and D. N. Sherman, "Traffic analysis of a ring switched data transmission system", *Bell System Technical Journal*, 50:9, November 1971, pp. 2947-2978. Reprinted in [Chu, 1976].
Delay estimates for a Pierce loop: analysis and simulation.

[Hayes & Sherman, 1972]
J. F. Hayes and D. N. Sherman, "A study of data multiplexing techniques and delay performance", *Bell System Technical Journal*, 51:9, November 1972, pp. 1983-2011. An "Extended abstract" with the same title appeared in *Proceedings of the National Telecommunications Conference (NTC '72)*, Houston, December 1972, paper 30D.
Analyses techniques for handling local terminal traffic to one processor/concentrator, using Polling, Aloha Random Access, and Ring. Computes resulting roundtrip delay.

[Heart, *et al.*, 1970]
F. E. Heart, R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA computer network", *AFIPS Conference Proceedings -- 1970 SJCC*, 36, June 1970, pp. 551-567. Reprinted in [Green & Lucky, 1975; Chu, 1976].
One of the early Arpanet papers.

[Heffron & Snow, 1977]
W. G. Heffron and N. E. Snow, "Transaction network system", *National Telecommunications Conference (NTC '77)*, Los Angeles, December 1977, paper 2:4.

[Heffron & Snow, 1978]
W. G. Heffron, Jr. and N. E. Snow, "Transaction network, telephones, and terminals: transaction network service", *Bell System Technical Journal*, 57:10, December 1978, pp. 3331-3347.
Message switching system for inquiry/response from a central host; polled control from a local switch.

[Hertzberg, *et al.*, 1979]
R. Y. Hertzberg, J. R. Schultz, J. F. Wanner, "The Promis network", *Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 87-111.
Uses 2-way CATV, with sequential polling from a controller.

[Hewlett-Packard, 1977a]
Hewlett-Packard Corporation, *Distributed Systems Networks*, October 1977.

[Hewlett-Packard, 1977b]
Hewlett-Packard Corporation, *Distributed Systems/1000, Technical Data*, October 1977.

[Hewlett-Packard, 1978a]
Hewlett-Packard Corporation, *Distributed Systems/1000*, March 1978.

[Hewlett-Packard, 1978b]
Hewlett-Packard Corporation, *DS/1000, Network Manager's Manual*, 1978.

[Hippert, 1970a]
R. O. Hippert, "A pulse-code-modulated transmission loop", *International Conference on Communications (ICC '70)*, San Francisco, June 1970, pp. 36-10 - 36-15.
IBM 2790 terminal system.

[Hippert, 1970b]
R. O. Hippert, "IBM 2790 Digital transmission loop", *IBM Journal of Research and Development*, 14:6, November 1970, pp. 662-667.

[Hobgood, 1976]
W. S. Hobgood, "The role of the Network Control Program in Systems Network Architecture", *IBM Systems Journal*, 15:1, 1976, pp. 39-52.

[Hopkins, 1977]
G. T. Hopkins, *A bus communications system*, Mitre Technical Report 3515, Mitre Corporation, Bedford, Mass., November 1977.

[Hopkins, 1979]
G. T. Hopkins, "Multimode communication on the Mitrenet", *Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 169-177.

[Hopper, 1978a]
A. Hopper, "Data ring at Computer Laboratory, University of Cambridge", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 11-16.

[Hopper, 1978b]
Andrew Hopper, *Local area computer communication network*, Technical Report no. 7 (PhD thesis), University of Cambridge Computer Laboratory, Cambridge, England, April 1978.

[Hopper & Wheeler, 1979]
A. Hopper and D. J. Wheeler, "Maintenance of ring communication systems", *IEEE Trans. on Comm.*, com-27:4, April 1979, pp. 760-761.
Extension of the Cambridge Ring work; have each station recompute parity, report an error to a monitoring station.

[Hsia, 1979]
P. Hsia, "A configurable distributed computing system," *1st Int. Conf. on Distributed Computing Systems*, Huntsville, October 1979, pp. 172-176.
Using a Hyperchannel to connect multiple VAX machines; uses an additional VAX bus for exchanging control messages.

[Huen, *et al.*, 1977]
W. Huen, P. Greene, R. Hochsprung, O. El-Dessouki, "A network computer for distributed processing", *15th IEEE Computer Society International Conference (COMPCON Fall '77)*, Washington, September 1977, pp. 326-330.
Ties together a group of LSI/11's to run a single program. Byte parallel ring.

[Hughes, 1977]
Hughes Aircraft, Facilities Management Systems -- Systems Description, Microelectronic Products Div., Facilities Management Systems Dept., Irvine, Cal., 1977.
A CATV system for facilities management, security, etc. 2-way cable; TDM.

[Hunt, 1978]
Bruce Hunt, "Ariel", unpublished talk, Stanford University, November 1978.

[IBM, 1975]
IBM, *IBM synchronous data link control, general information*, 2nd edition, May 1975.

[IBM, 1978a]
IBM, *An introduction to the IBM 8100 information system*, publication GA27-2875, 1978. (Second edition, March 1979.)
The 8100 has provision for an SDLC loop-mode link for connecting peripherals. Can have a "directly attached loop" or use a communication line to a "data link attached loop."

[IBM, 1978b]
IBM, *IBM 8100 information system configurator*, publication GA27-2876, October 1978.

[IBM, 1978c]
IBM, *IBM 8100 information system loop installation manual -- physical planning*, publication GA27-2878, October 1978.
Includes photographs and detailed description of equipment used to construct SDLC loop systems for the 8100.

[IEEE, 1975]
IEEE Standard digital interface for programmable instrumentation, IEEE Standard 488-1975, IEEE, New York, April 1975; revised 1978.

[IEEE, 1979]
IEEE, *4th Conference on local computer networks*, Minneapolis, Minnesota, October 1979.

[IFIP, 1979]
IFIP Working Group 6.4 on Local Computer Networks, *Statement of aims and scope*, June 1979.

[Innes & Alty, 1975]
D. R. Innes and J. L. Alty, "An intra university network", *4th Data Communications Symposium*, ACM and IEEE, Quebec City, October 1975, pp. 1-8 - 1-13.
Star configuration: mini-computers attached to a single Support Computer, which in turn interfaces to a large-scale machine. Byte parallel links to the Support Computer, up to about 250 kbps.

[Iwama, *et al.*, 1978]
K. Iwama, Y. Kambayashi, and S. Yajima, "Computer communication interfaces based on two-input/output-pair automata and their implementation in the Labolink network", *Fourth International Conference on Computer Communication (ICCC)*, Kyoto, September 1978.

[Jafari, 1977]
H. Jafari, *A new loop structure for distributed microcomputing systems*, PhD Dissertation, Oregon State University, Corvallis, December 1977, available from University Microfilms.

[Jafari & Lewis, 1977]
H. Jafari and T. Lewis, "A new loop structure for distributed microcomputing systems", *Proceedings of the First Annual Rocky Mountain Symposium on Microcomputers: Systems, Software and Architecture*, 1977, pp. 121-141.

[Jafari, *et al.*, 1978a]
H. Jafari, J. Spragins, and T. Lewis "A new modular loop architecture for distributed computer systems", *Trends and Applications 1978: Distributed Processing*, NBS/IEEE(DC) Symposium, 1978, pp. 72-77.

[Jafari, *et al.*, 1978b]
H. Jafari, T. Lewis, and J. Spragins "A new ring-structured microcomputer network", *Fourth International Conference on Computer Communication (ICCC)*, Kyoto, Japan, September 1978.
Really a loop, with a *loop controller*. Two channels, one for control and one for data; negotiate on control channel to set switches at intermediate nodes on the data ring.

[Jensen, 1974]
E. Douglas Jensen, "A mini-multicomputer for real-time control", *9th Annual IEEE Computer Society Int. Conf. (Compcon Fall '74)*, Washington, September 1974, pp 245-248.
Multiple minicomputers, connected with a set of global buses; 16 bits of parallel data.

[Jensen, 1975]
E. Douglas Jensen, "A distributed function computer for real-time control", *2nd Annual Symposium on Computer Architecture*, ACM SigArch, January 1975; proceedings published as *Computer Architecture News*, December 1974.
Proposal for a Modular Computer System (MCS): mini computers with global buses, 16 data bits. Each host passes control on a bus Synch Line; causes all hosts to increment through a bit vector indicating which slots are theirs to use. Complicated bus interface logic.

[Jensen, 1978]
E. Douglas Jensen, "The Honeywell experimental distributed processor -- an overview", *Computer*, January 1978, pp. 28-38.
Multiple hosts connected with a serial bus. Vector Driven Proportional Access (VDPA): special bus signal used to transfer control; all hosts index through a bit vector indicating when a host gets control. 5-processor prototype built; complex bus interface units.

[Kanakia & Thomasian, 1979a]
H. Kanakia and A. Thomasian, "A comparative study of access control mechanisms in loop networks," *4th Conf. on Local Computer Networks*, Minneapolis, October 1979, pp. 104-110.

[Kanakia & Thomasian, 1979b]
H. Kanakia and A. Thomasian, "Performance modeling of loop networks with priority classes," *4th Conf. on Local Computer Networks*, Minneapolis, October 1979, pp. 75-81.

[Kasson, 1979]
James M. Kasson, "Survey of digital PBX design", *IEEE Trans. on Comm.*, com-27:7, July 1979, pp. 1118-1124.
General discussion of digital PBX's; does not specifically focus on techniques for sending data through them.

[Kaye, 1972]
A. R. Kaye, "Analysis of a distributed control loop for data transmission", *Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 47-58.
Analysis of a Newhall-style, control passing system.

[Kitazawa, 1976]
Shigeyoshi Kitazawa, *Development of an in-house computer network Kuipnet*, Department of Information Science (thesis), Kyoto University, December 1976.
Modeled after the Arpanet, but a simple star with a single IMP.

[Kitazawa & Sakai, 1978]
Shigeyoshi Kitazawa and Toshiyuki Sakai, "Performance evaluation of the Kuipnet computer network", *Computer Communications*, 1:3, June 1978, pp. 149-155.
Provides some real performance measurements for this modest star-shaped packet switching system.

[Kleinrock & Naylor, 1974]
L. Kleinrock and W. E. Naylor, "On measured behavior of the ARPA network", *AFIPS Conference Proceedings -- NCC '74*, 43, pp. 767-780.
Includes lots of Arpanet measurements, as well as a brief discussion of *incestuous* traffic -- traffic between two local hosts transmitted to the same Imp.

[Knoblock, *et al.*, 1975]
D. E. Knoblock, D. C. Loughry, and C. A. Vissers, "Insight into interfacing", *IEEE Spectrum*, May 1975, pp. 50-57.
Report on proposed IEEE 488 instrumentation standard: parallel bus with a controller, for up to 15 devices.

AN ANNOTATED BIBLIOGRAPHY ON LOCAL COMPUTER NETWORKS

[Konheim, 1972]
A. G. Konheim, "Service epochs in a loop system", *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 125-143.
Loop system with a central station and N terminals.

[Konheim, 1976]
A. G. Konheim, "Chaining in a loop system", *IEEE Trans. on Comm.*, com-24:2, February 1976, pp. 203-210.
Loop with central control and chaining, or hub-polling.

[Kropfl, 1972]
W. J. Kropfl, "An experimental data block switching system", *Bell System Technical Journal*, 51:6, July-August 1972, pp. 1147-1165.
Originally written in February 1971; describes an implementation of the Pierce loop augmented with a mechanism for "hog prevention." Used TI technology for a single loop, with one A box (controller) and 2 B boxes.

[Kuhns & Shoquist, 1979]
Richard C. Kuhns and Marc C. Shoquist, "A serial data bus system for local processing networks", *18th IEEE Computer Society International Conference (COMPCON Spring '79)*, San Francisco, February 1979, pp. 266-271.
10 Mbps Triax cable, separate control and data lines, central control.

[LaBarre, 1978]
C. E. LaBarre, *Analytic and simulation results for CSMA contention protocols*, Mitre Corp., MTR-3672, November 1978.
Nice comparision of analytic and simulation results for several CSMA techniques.

[Labonte, 1973]
Robert C. Labonte, "Developing a wired nation -- a general purpose digital communications system for operation on a conventional CATV system", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, San Francisco, February 1973, pp. 85-88.

[Lam, 1979]
Simon S. Lam, "A study of the CSMA protocol in local networks", *Proc. of the 4th Berkeley Conference on Distributed Data Mangement and Computer Networks*, San Francisco, August 1979, pp. 141-154.
Analysis of a slotted CSMA scheme.

[Lam, 1980]
Simon S. Lam, *A packet network architecture for local interconnection*, University of Texas at Austin, Technical Report TR-130, February 1980.
Proposal for an Ethernet-style system, augmented with a priority access scheme (similar to the Hyperchannel approach). Uses the Zilog SIO, run at around 500 Kbps.

[Lampson & Simonyi, 1979]
Butler Lampson and Charles Simonyi, personal communication, Xerox Palo Alto Research Center, July 1979
Proposal for a high performance local network, up to 50 Mbps, using an ECL micro-Imp.

[Lancaster & Garodnick, 1973]
Paul Lancaster and Joseph Garodnick, "CATV environment for data communications", *National Telecommunications Conference (NTC '73)*, Atlanta, November 1973, paper 38C.

[Laurer & Skatrud, 1977]
G. J. Laurer and R. O. Skatrud, "Automatic loop reconfigurator", *IBM Technical Disclosure Bulletin*, 19:10, March 1977, pp. 3824-3828.
Scheme to reconfigure a broken loop as two half-duplex, multi-drop lines.

[Lebetoulle, *et al.*, 1975]
J. Lebetoulle, E. G. Manning, and R. W. Peebles, "A homogeneous computer network -- analysis and simulation", *Computer Networks*, 1977, pp. 225-240. (An earlier report appeared as Computer Communications Networks Group Report E-30, University of Waterloo, Waterloo, Ontario, Canada, January 1975.)
Modeled a very specialized environment: identical hosts connected to a Newhall-Farmer style loop, doing restricted, transaction interactions over a shared data base. Limited protocols. Initially, two PDP-11's as user hosts. Note: "Minet" is the name of the architecture for handling distributed transaction processing; there have been no reports of actual experience with any local network implementations.

[LeBoss, 1980]
B. LeBoss, "Network links units, includes mass store," *Electronics*, January 31, 1980, pp. 43-44.
Discusses the Nestar systems for connecting personal computers.

[Lee & Pohm, 1978]
Chong C. Lee and Arthur V. Pohm, "Interface processor for high speed recirculating data network", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, September 1978.
Proposal for ISLInet, a T1 ring with 32 circulating slots, 16 bytes per slot.

[Le Lann, 1977]
G. Le Lann, "Distributed systems -- towards a formal approach", *Information Processing 77 (Proc. of IFIP 77)*, Toronto, August 1977, pp. 155-160.
See esp. section 4, techniques which might be used to regenerate a lost control token in a token-passing ring.

[Lennon, *et al.*, 1973]
William J. Lennon, Ronald C. Barrett, and John T. Spies, "A mini-computer research network", *7th Annual IEEE Computer Society International Conference (COMPCON 73)*, San Francisco, February 1973, pp. 151-154.

[Lennon, 1974]
William J. Lennon, "A mini-computer network for support of real time research", *ACM 1974 Annual Conf. (ACM '74)*, San Diego, November 1974, pp. 595-604.

[Lennon, 1975]
William J. Lennon, "A user oriented mini-computer network", *11th IEEE Computer Society International Conference (COMPON Fall '75)*, Washington, September 1975, 133-136.
Simple star system, nodes linked to central point; looks like a paper tape device to users' machines.

[Lewis, 1977]
Fred V. Lewis, "Use of a rotating-master loop network as a high integrity intersystem data link", *Proceedings of the National Electronics Conference*, 31, Chicago, October 1977, p. 30.
One-page proposal for a ring using SDLC control frames, with "ring-master" status being passed around the ring.

[Lidinsky, 1976]
William P. Lidinsky, "The Argonne Intra-Laboratory Network", *Proc. of the [1st] Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Laboratory, Berkeley, California, May 1976, pp. 263-275.
Slotted store-&-forward, 50 Kbps lines, packet switching through Interface Processing Units (IPUs, like Imps) containing dual microprocessors.

[Lin, 1978]
Kuang-Shin Lin, "Design of a packet-switched micro-subnetwork", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, September 1978, pp. 184-193.
A small network with 4 hosts and 4 "micro-Imps."

[Liu, 1978]
Ming T. Liu, "Distributed Loop Computer Networks", in M. C. Yovits, ed., *Advances in Computers, Volume 17*, Academic Press, 1978, pp. 163-221.

[Liu & Reames, 1975]
Ming T. Liu and Cecil C. Reames, "The design of the Distributed Loop Computer Network", *International Computer Symposium 1975*, Taipei, August 1975, pp. 273-283 (vol. 1).

[Liu & Reames, 1977]
Ming T. Liu and Cecil C. Reames, "Message communication protocol and operating system design for the distributed loop computer network (DLCN)", *4th Annual Symposium on Computer Architecture*, March 1977, pp. 193-200.

[Liu, *et al.*, 1977]
M. T. Liu, G. Babic, and R. Pardo, "Traffic analysis of the distributed loop computer network (DLCN)", *National Telecommunications Conference (NTC '77)*, December 1977, paper 31:5.

[Liu, *et al.*, 1979a]
J. W. S. Liu, I. Suwa, R. Stepp, S. M. Hinojosa, and T. Utsuqi, "A fail-safe distributed local network for data communications," *AFIPS Conference Proceedings (NCC '79)*, 48, New York, June 1979, pp. 917-925.
Proposal for a richly connected, high-reliability store-and-forward system; sets up virtual calls through the switches, however.

[Liu, *et al.*, 1979b]
M. T. Liu, R. Pardo, D. Tsay, J. J. Wolf, B. W. Weide, and C. Chou, "System design of the distributed double-loop computer network (DDLCN)," *1st Int. Conf. on Distributed Computing Systems*, Huntsville, October 1979, pp. 95-105.
Successor to the previous system, DLCN.

[Loomis, 1973(?)]
D. C. Loomis, *Ring communication protocols*, Technical Report #26, Dept. of Information and Computer Science, UC Irvine, undated (but listed elsewhere as January 1973).
Describes use of a control token, passed around to control the ring.

[Loughry, 1972]
Donald C. Loughry, "A common digital interface for programmable instruments: the evolution of a system", *HP Journal*, 24:2, October 1972, pp. 8-11.
Background on development of HP interface bus; see also [Nelson & Ricci, 1972].

[Loughry & Allen, 1978]
Donald C. Loughry and Mark S. Allen, "IEEE standard 488 and microprocessor synergism", *Proc. of the IEEE*, 66:2, February 1978, pp. 162-172.
Good discussion of the IEEE 488 instrumentation bus standard; includes a lengthy bibliography.

[Loveland, 1979]
Richard A. Loveland, "Putting Decnet into perspective", *Datamation*, March 1979, pp. 109-114.
Recites some of the Decnet history, including lack of routing in Phase I and Phase II.

[Loveland & Stein, 1979]
Richard A. Loveland and Charles W. Stein, "How Decnet's communications software works", *Data Communications*, January 1979.
Lots of prose, not much hard information.

[Luczak, 1978]
E. C. Luczak, "Global bus computer communications techniques," *Computer Networking Symposium*, IEEE(CS)/NBS, Gaithersburg, December 1978, pp. 58-70.
Good summary and comparison of alternative bus access schemes.

[Lyle & Farber, 1976]
Michael R. Lyle and David J. Farber, "Transmission systems tradeoffs in ring-structured digital systems", *Proceedings of the [1st] Berkeley Workshop on Distributed Data Management and Computer Networks*, Lawrence Berkeley Laboratory, Berkeley, May 1976.
Early discussion of design considerations for the Local Network Interface (LNI), including a single chip, LSI version.

[MacLaren, 1978]
M. D. MacLaren, "Arbitarting a multi-drop line: uniform p-persistence," *17th IEEE Comp. Soc. Int. Conf. (Compcon Fall '78)*, Washington, September 1978, pp. 110-114.

[Majithia & Dube, 1976]
J. C. Majithia and J. D. Dube, "Results for a loop network with a hybrid message-handling protocol", *Proc. IEE*, 123:8, August 1976, pp. 775-776.
Short note describing simulation of a hypothetical full-duplex ring, with different priority classes.

[Majithia & Dube, 1977]
J. C. Majithia and J. D. Dube, "Analysis and simulation of message-switched loop data networks", *Proc. IEE*, 124:3, March 1977, pp. 193-197.
Analysis and simulation of a full-duplex ring.

[Manning, 1972]
Eric G. Manning, "Newhall loops and programmable TDM -- two facets of Canadian research in computer communications", *First Int. Conference on Computer Communication (ICCC)*, Washington, October 1972, pp. 338-342.
Proposal to be a Newhall loop to connect TIP-like nodes. Brief description of a 2-node Newhall loop implemented at the University of Toronto (to Waterloo): METANET. Notes that passing of control from one node to the next takes 2 msec.

[McEnroe, *et al.*, 1975]
P. V. McEnroe, H. T. Huth, E. A. Moore, and W. W. Morris, "Overview of the supermarket system and the retail store system", *IBM Systems Journal*, 14:1, 1975.
Loop system for terminals.

[McFadyen, 1976]
J. H. McFadyen, "Systems Network Architecture: An overview", *IBM Systems Journal*, 15:1, 1976, pp. 4-23.

[McKenzie, 1979]
Alex McKenzie, personal communication, BBN, July 1979.
Described several scaled-down or derivative versions of the Arpanet.

[McQuillan, 1978]
John M. McQuillan, *Understanding the new local network technologies*, BBN Report 3927, September 1978.

[McQuillan & Walden, 1977]
J. M. McQuillan and D. C. Walden, "The ARPA Network design decisions", *Computer Networks*, 1:5, August 1977, pp. 243-289.
This is an expanded version of [Crowther, *et al.*, 1975]: it is one of the best all-around papers on the Arpanet, explaining many aspects of the design. Includes a short discussion of problems associated with network front end machines (esp. sections 3.3.2 and 5.2.2).

[Meisner & Rosenthal, 1979]
N. B. Meisner and R. Rosenthal, eds., *Proc. of the Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979.

AN ANNOTATED BIBLIOGRAPHY ON LOCAL COMPUTER NETWORKS

[Meisner, *et al.*, 1977a]
Norman B. Meisner, Joshua L. Segal, and Malcolm Y. Tanigawa, "Dual-mode slotted TDMA digital bus", *5th Data Communications Symposium*, Snowbird, Utah, September 1977, 5-14 - 5-18.

[Meisner, *et al.*, 1977b]
Norman B. Meisner, David G. Willard, and Gregory T. Hopkins, "Time division digital bus techniques implemented on coaxial cable", *Computer Networking Symposium*, IEEE Computer Society and NBS, Gaithersburg, Maryland, December 1977, pp. 112-117.

[Mendicino, 1970]
Samuel F. Mendicino, "Octopus: The Lawrence Radiation Laboratory Network", in Rustin, ed., *Computer Networks (Courant Computer Science Symposium 3, November-December 1970)*, Prentice-Hall, New York, 1972, pp. 95-110.

[Mendicino & Sutherland, 1973]
Sam F. Mendicino and George G. Sutherland, "Performance measurements in LLL Octopus computer network", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, San Francisco, February 1973, pp. 109-112.

[Metcalfe & Boggs, 1976]
Robert M. Metcalfe and David R. Boggs, "Ethernet: Distributed packet switching for local computer networks", *CACM*, 19:7, July 1976, pp. 395-404. (Earlier version issued as Xerox Parc Computer Science Report CSL-75-7.) Reprinted in Abrams, *et al.*, eds., *Computer Networks: A Tutorial (Revised 1978)*, IEEE, 1978.

[Metcalfe, *et al.*, 1977]
Robert M. Metcalfe, David R. Boggs, Charles P. Thacker, and Butler W. Lampson, *Multipoint data communication system with collision detection*, US Patent no. 4,063,220, December 1977.

[MIL-STD-1553B, 1978]
MIL-STD-1553B: Military Standard, Aircraft internal time division command/response multiplex data bus, Aeronautic Systems Division, Wright-Patterson Air Force Base, Sept. 1978.
Military standard for digital comm. inside an aircraft. Serial bus, 1 Mbps, Manchester encoding, controlled from a centralized bus controller. Up to 32 devices per network, 32 words per block.

[Miller, 1978]
Walt Miller, "Interconnecting local networks", *Third Conference on Local Computer Networks*, U. of Minn., Minneapolis, October 1978.
NSC/BNR project, interconnect Hyperchannel systems via T3 microwave.

[Mockapetris, 1978]
Paul V. Mockapetris, *Design considerations for the ARPA LNI name table*, Technical Report 92, Department of Information and Computer Science, UC Irvine, Revised May 1978.

[Mockapetris & Farber, 1977]
Paul V. Mockapetris and David J. Farber, "Experience with the Distributed Computer System (DCS)", Technical Report 116, Department of Information and Computer Science, UC Irvine, 1977.

[Mockapetris, *et al.*, 1977]
Paul V. Mockapetris, Michael R. Lyle, and David J. Farber, "On the design of local network interfaces", *Information Processing '77 (Proc. of IFIP '77)*, Toronto, August 1977, pp. 427-430.

[Mok & Ward, 1979]
A. K. Mok and S. A. Ward, "Distributed broadcast channel access," *Computer Networks*, 3:5, November 1979, pp. 327-335.
Priority schemes for accessing a parallel bus -- see also [Nesnevich and Strasbourger, 1974]; generally aimed at back plane buses.

[Moller, 1978]
Chris Moller, "A simple data bus for low data rates," *Electronic Engineering*, 50:606, May 1978, pp. 41-43.
A simple serial bus with a bus controller; for process control/instrumentation applications.

[Moran, 1975]
D. M. Moran, "Memory multiplexer data link, an intermodular network", *IEEE Electronics and Aerospace Systems Convention (Eascon '75)*, Washinton, September 1975, paper 16?.
MMDL connects processors and memories in a multi-processor; uses a form of "distributed polling" around the ring.

[Moran & Starkson, 1975]
D. M. Moran and R. O. Starkson, "A hybrid communications switching system", *Electronic Components Conference*, 1975, pp. 30-36.
Describes some components later used in a centralized circuit switch, [Sperry-Univac, 1977].

[Morling, *et al.*, 1978]
R. C. S. Morling, G. Neri, G. D. Cain, E. Faldella, T. Salmon, and D. J. Stedham, "The Minitet inter-node control protocol", *Symposium on Computer Network Protocols*, Liege, 1978, paper B4.

[Moulton & Sander, 1977]
P. D. Moulton and R. C. Sander, "Another look at SNA", *Datamation*, March 1977, pp. 74-82.

[Moura, *et al.*, 1979]
J. A. B. Moura, J. A. Field, J. W. Wong, "Evaluation of collison control algorithms in Ethernets," *6th Data Comm. Sym.*, Pacific Grove, November 1979, pp. 82-86.
Includes simulation models of several schemes; differences in underlying assumptions make it difficut to compare with the actual Ethernet implementation.

[Naylor, 1978]
J. C. Naylor, Jr., "Data bus design concepts, issues and prospects", *IEEE Electronics and Aerospace Systems Convention (Eascon '78)*, Arlington, Virginia, September 1978, pp. 34-39.

[NBS, 1977]
"NBS experimenting with 'Ethernet' packet switching", *Data Communications*, February 1977, p. 20.

[NBS, 1978a]
National Bureau of Standards, *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978.

[NBS, 1978b]
"NBS to construct local packet network", *Data Communications*, November 1978, p. 22.

[NBS, 1980]
"NBS uncaging local network", *Data Communications*, January 1980, pp. 31-32.
Describes expanded operation of NBS's Ethernet-like system.

[Needham, 1979]
Roger M. Needham, "Systems aspects of the Cambridge Ring", *7th Symposium on Operating System Principles*, Pacific Grove, December 1979, pp. 82-85.

[Nelson & Gordon, 1978]
David L. Nelson and Robert L. Gordon, "Computer cells -- a network architecture for data flow computing", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, September 1978, pp. 296-301.
This system includes a communications ring connecting Prime P400 computers, running at 10 Mbps.

[Nelson & Ricci, 1972]
Gerald E. Nelson and David W. Ricci, "A practical interface system for electronic instruments", *HP Journal*, 24:2, October 1972, pp. 2-7.
Early report on HP interface bus; later evolved into IEEE 488.

[Nessett, 1978]
Dan Nessett, "Protocols for buffer space allocation in CSMA broadcast networks with intelligent interfaces", *Third Conference on Local Computer Networking*, U. of Minn., Minneapolis, October 1978.
Some bus networks (such as the Hyperchannel) provide packet buffers as part of the interface; discusses ways to manage buffer allocation in these units.

[NESTAR, 1979(?)]
NESTAR Systems Inc., *An overview of the Cluster/One system* (product brochure), Palo Alto, California, 1979(?).
Cluster/One is a file server for personal computers; the "Queen" (a Pet computer) runs a set of "drones" through a parallel bus communications link. Max of 15 drones per channel; can be Pet, TRS-80, or Apple II.

[NESTAR, 1980]
NESTAR Systems Inc., *Nestar announces the Clustershared microcomputer system* (press release), Palo Alto, Cal., January 1980.
The Cluster/One "Model A": uses an Apple II as the controller; services up to 64 hosts (only Apple II).

[Newhall & Venetsanopoulos, 1971]
E. E. Newhall, and A. N. Venetsanopoulos, "Computer communications -- representative systems", *Information Processing 71 (Proc. of IFIP 71)*, Ljubljana, August 1971, pp. 545-552.
Summaries of the Arpanet, the Collins C-System TDM loop, and the Farmer/Newhall loop.

[Nisnevich & Strasbourger, 1974]
L. Nisnevich and E. Strasbourger, "Decentralized priority control in data communication," *Proc. of the 2nd Annual Symposium on Computer Architecture/Computer Architecture News*, December 1974, pp. 1-6.
Priority schemes for accessing a shared bus -- generally in the context of a parallel processor.

[Noguchi & Shiratori, 1976]
S. Noguchi and N. Shiratori, "Fundamental characteristics of loop computer network", *International Switching Symposium*, Kyoto, October 1976, paper 133-2.
Analytic treatment of a loop; augmentation with "by-pass" links.

[Noguchi, *et al.*, 1974]
S. Noguchi, N. Shiratori, K. Teruya, and J. Oizumi, "On characteristics of loop computer network", *Computer Nets: A Supplement to the Proceedings of the 7th Hawaii International Conference on System Sciences*, January 1974, pp. 24-26.
A brief consideration of a ring structure where collisions may take place, Aloha-style.

[NSC, 1976]
Network Systems Corp., (untitled paper), *Conference on Experiments in New Approaches to Local Computer Networking*, U. of Minn., St. Paul, September 1976.

[Oh & Liu, 1977]
Y. Oh and M. T. Liu, "Interface design for distributed control loop networks", *National Telecommunications Conference (NTC '77)*, December 1977, paper 31:4.
Proposed structure for a DCLN interface; lots of hardware, introduces delay at each node.

[Ohnsorge & Schenkel, 1974]
H. Ohnsorge and K. D. Schenkel, "An integrated communication system with fully decentralized switching", *IEEE Trans. on Comm.*, com-22:9, September 1974, pp. 1292-1296.
Proposal for a bus system for voice and data, using distinct transmit and receive tree structures. Stations individually set up TDM channels to carry voice or data; the system does depend upon a central clock.

[Okuda, *et al.*, 1978]
N. Okuda, T. Kunikyo, and T. Kaji, "Ring Centry Bus -- an experimental high speed channel for computer communications", *Fourth International Conference on Computer Communication (ICCC)*, Kyoto, September 1978.
Control station generates timing pulses of the ring, and clears any packet which circulates 256 times (!); both a data and control loop, using fiber optics. 100 Mbps. TDM, 8 channels (packets) circulating within exactly 1 frame; if channel is empty, can grab it, and reuse it repeatedly.

[Onoe, *et al.*, 1978]
Morio Onoe, Yasuhiko Yasuda, and Mitsuru Ishizuka, "A random access packet communication system with priority function -- Priority Ethernet", *National Convention of the Information Processing Society of Japan*, August 1978, paper no. 3A-1.

[Ornburn & McKeown, 1975]
F. Helmuth Ornburn and David M. McKeown, Jr., "A packet switching network for minicomputers", *11th IEEE Computer Society International Conference (COMPCON Fall '75)*, Washington, September 1975.
Parallel bus with an arbiter; micro-imps between the hosts and the bus, to minimize host changes. Sort of like a super Unibus, can read and write to other units.

[Owens, 1973]
Jerry L. Owens, "A user's view of the Lawrence Livermore Laboratory's computer networks", *7th Annual IEEE Computer Society International Conference (COMPCON '73)*, San Francisco, February 1973, pp. 75-78.

AN ANNOTATED BIBLIOGRAPHY ON LOCAL COMPUTER NETWORKS

[Pardo, *et al.*, 1977]
Roberto Pardo, Ming T. Liu, and Gojko A. Babic, "Distributed services in computer networks: designing the distributed loop date [sic] base system", *Computer Networking Symposium*, IEEE Computer Society and NBS, Gaithersburg, December 1977, pp. 60-65.

[Passafiume & Wecker, 1977]
Joseph J. Passafiume and Stuart Wecker, "Distributed file access in Decnet", *Proceedings of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977, pp. 114-129.

[Patton & Franck, 1976]
Peter C. Patton and Abe Franck, eds., *Conference on Experiments in New Approaches to Local Computer Networking*, U. of Minn., St. Paul, September 1976.
Mainly focused on techniques related to "large scale" hosts. This first conference was co-sponsored with NSC, and many of the papers are from potential users of the Hyperchannel. Only selected papers have been included in this bibliography.

[Patton & Franck, 1977]
Peter C. Patton and Abe Franck, eds., *Conference on "A Second Look at Local Computer Networking"*, U. of Minn., Minneapolis, October 1977.

[Patton & Franck, 1978]
Peter C. Patton and Abe Franck, eds., *Third Conference on Local Computer Networking*, U. of Minn., Minneapolis, October 1978.

[Paulish, 1979]
D. J. Paulish, "The exploratory system control model multi-loop network," *AFIPS Conference Proceedings (NCC '79)*, 48, New York, June 1979, pp. 935-940.
Newhall-style token-passing loop, running at 1 Mbps; multiple loops connected with gateways. System implemented as a laboratory prototype; each loop implemented within a single cabinet.

[Pehrson, 1973]
David L. Pehrson, "Interfacing and data concentration", in Abramson and Kuo, eds., *Computer-Communication Networks*, Prentice-Hall, 1973.
Esp. section 6.8, an extended example treating the Octopus network.

[Penney & Baghdadi, 1979a]
B. K. Penney and A. A. Baghdadi, "Survey of computer communication loop networks: part 1," *Computer Communications*, 2:4, August 1979, pp. 165-180.

[Penney & Baghdadi, 1979b]
B. K. Penney and A. A. Baghdadi, "Survey of computer communication loop networks: part 2," *Computer Communications*, 2:5, October 1979, pp. 224-241.

[Pfleger, 1979]
Doina Pfleger, "Availability of loop computer networks," *4th Conf. on Local Computer Networks*, Minneapolis, October 1979, pp. 119-130.
Analysis of availability with multi-stage loop configurations.

[Pierce, 1972a]
J. R. Pierce, "How far can data loops go?", *IEEE Trans. on Comm.*, com-20, June 1972, pp. 527-530. Reprinted in [Green & Lucky, 1975; Chu, 1976].

[Pierce, 1972b]
J. R. Pierce, "Network for block switching of data", *Bell System Technical Journal*, 51:6, July-August 1972, pp. 1133-1145.
Originally written in 1970. Terminals connected to loops; loops may then be interconnected. Uses regular digital lines (T1), without special modulation (unlike Newhall). Each loop requires a special "A" box, for loop timing, etc.

"The data network has been deliberately kept very simple. It is multiprocessing with a vengeance."

[Pierce, *et al.*, 1971]
J. R. Pierce, C. H. Coker, and W. J. Kropfl, "An experiment in addressed block data transmission around a loop", *IEEE International Convention Record*, March 1971, pp. 222-223.
Pierce's first paper on the subject. Describes possible hierarchy of loops, connecting the entire country.

[Pogran & Reed, 1978]
K. T. Pogran and D. P. Reed, "The MIT Laboratory for Computer Science Network", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 20-22.

[Pohm, *et al.*, 1979]
A. V. Pohm, J. A. Davis, S. Christiansen, G. D. Bridges, R. E. Horton, "A local network of mini and microcomputers for experiment support," *4th Conf. on Local Computer Networks*, Minneapolis, October 1979, pp. 91-96.
TDM ring: 256 slots*8bit/slot; each destination has an assigned slot. Can support host-to-host traffic, but mainly describes use to support terminal traffic from Commodore Pets to a PDP-11.

[Popescu-Zeletin, 1979]
R. Popescu-Zeletin, "The data access and transfer support in a local heterogeneous network (HMINET)," *6th Data Comm. Sym.*, Pacific Grove, November 1979, pp. 147-152.
A star system, with a single packet switch in the center. Joint project of HMI and Siemens.

[Popescu-Zeletin, *et al.*, 1979]
R. Popescu-Zeletin, D. Baum, and B. Butscher, "A local computer network in a research environment: the HMINET," *Computer Networking Symposium*, IEEE(CS)/NBS, Gaithersburg, December 1979, pp. 158-163.

[Potter & Paulish, 1979]
M. R. Potter and D. J. Paulish, "The modular system control development model (MSCDM)," *1st Int. Conf. on Distributed Computing Systems*, Huntsville, October 1979, pp. 514-519.
Application for the Burroughs ESM loop.

[Potvin, *et al.*, 1971]
J. N. Potvin, P. Chenevert, K. C. Smith and P. Boulton, "Star-Ring: a computer intercommunication and I/O system", *Information Processing 71 (Proc. of IFIP 71)*, Ljubljana, August 1971.
Central high-speed parallel ring (option for buses radiating out from a node); daisy chain of control around the ring; address the destination explicitly on 1 of n address lines.

[Prime, 1979]
Prime Computer Inc., *The Primenet Guide*, Document IDR3710, June 1979.
Includes a description of the Primenet Node Controller Ring (PNC): token passing ring, 8 Mbps, up to 8 hosts per ring.

[Raimondi, *et al.*, 1976]
D. L. Raimondi, H. M. Gladney, G. Hochweller, R. W. Martin, and L. L. Spencer, "LABS/1 -- a distributed real-time operating system", *IBM Systems Journal*, 15:1, 1976, pp. 81-101.
Originally a laboratory automation system, connecting a group of System/1's in a star to a 360 or 370. Central host used for program development, data analysis, etc.

[Rawson, 1979a]
E. G. Rawson, "Optical fibers for local computer networks", *Digest of Topical Meeting on Optical Fiber Communication*, Washington, March 1979, pp. 60-64.

[Rawson, 1979b]
E. G. Rawson, "Application of fiber optics to local networks", *Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 155-168.

[Rawson & Metcalfe, 1978]
Eric Rawson and Robert M. Metcalfe, "Fibernet: multimode optical fibers for local computer networks", *IEEE Trans. on Comm.*, 26:7, July 1978, pp. 983-990.
Compares alternative architectures for local nets using fiber optics; reports on Fibernet, a transmissive star coupler formed using a mixing rod. Good collection of references on work in this area.

[Rawson & Nafarrate, 1978]
Eric Rawson and Antonio B. Nafarrate, "Star couplers using fused biconically tapered multimode fibres", *Electronic Letters*, 14:9, April 27, 1978, pp. 274-275.
Reports on 19-channel transmissive star couplers (a la Fibernet), but now formed with thermal fusing.

[Rawson, *et al.*, 1978]
E. G. Rawson, R. M. Metcalfe, R. E. Norton, A. B. Nafarrate, and D. Cronshaw, "Fibernet: A fiber optic computer network experiment", *Proceedings of the Fourth European Conference on Optical Communication (4th ECOC)*, Genova, Italy, September 1978.

[Reames & Liu, 1975]
Cecil C. Reames and Ming T. Liu, "A loop network for simultaneous transmission of variable-length messages", *2nd Annual Symposium on Computer Architecture*, ACM SigArch, January 1975, pp. 7-12.

[Reames & Liu, 1976]
Cecil C. Reames and Ming T. Liu, "Design and simulation of the Distributed Loop Computer Network (DLCN)", *3rd Annual Symposium on Computer Architecture*, ACM SigArch, January 1976, 124-129.

[Richardson & Yu, 1979]
R. G. Richardson and L. W. Yu, "The effect of protocol on the response time of loop structures for data communications", *Computer Networks*, 3, 1979, pp. 57-66.
Models inquiry/response traffic for N terminals and 1 host.

[Robilland, 1974]
P. N. Robilland, "An analysis of a loop switching system with multirank buffers based on the Markov process", *IEEE Trans. on Comm.*, 22:11, November 1974, pp. 1772-1778.
Considers a Farmer/Newhall loop: buffered I/O terminals, a loop, and a loop supervisor to provide clock and failure control.

[Rodgers, 1977]
John C. Rodgers, "Computer networking with a data bus", *16th Annual Technical Symposium*, ACM(DC) and NBS, Gaithersburg, Maryland, June 1977, pp. 45-50.
Plan to use an NSC Hyperchannel; run the 4 trunks at different rates.

[Rosen, 1980]
B. Rosen, "PERQ Packet Stream Network," *20th IEEE Comp. Soc. Int. Conf. (Compcon Spring '80)*, San Francisco, February 1980, pp. 317-317-a.
Brief overview of the PERQ networking system: similar to the Ethernet (but without collision detection), running at 10 Mbps.

[Rosen & Steele, 1973]
Saul Rosen and John M. Steele, "A local computer network", *7th Annual IEEE Computer Society International Conference (COMPCON 73)*, San Francisco, February 1973, pp. 129-132.
Star configuration, terminals and small hosts access a CDC 6500, 9.6 Kbps lines.

[Rowe, 1975]
L. A. Rowe, *The Distributed Computing Operating System*, DCS Technical Report #66, June 1975.
A nice overview of the system, and discussion of software. Includes a tiny bit of performance information.

[Rowe & Birman, 1979]
Lawrence A. Rowe and Kenneth P. Birman, "Network support for a distributed data base system", *Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks*, San Francisco, August 1979, pp. 337-351.
The "Cosmos" is a local network being built to support research on distributed data bases; the network is being built with the "LNI" ring interfaced originally developed at UC Irvine [Mockapetris, *et al.*, 1977].

[Rowe, *et al.*, 1973]
Lawrence A. Rowe, Marsha D. Hopwood, David J. Farber, "Software methods for achieving fail-soft behavior in the Distributed Computing System", *IEEE Symposium on Computer Software Reliability*, April 1973.

[Rudin, 1974]
H. Rudin, "The design of digital communications systems: can traffic theory help?", *International Zurich Seminar on Digital Communication*, Zurich, March 1974, paper D5.
General discussion of loops, and summary of analytic results.

[Saito, 1978]
T. Saito, "A subscriber carrier system based upon frame addressing system", *IEEE Trans. on Comm.*, com-26:8, August 1978, pp. 1287-1295.
Loop system for carrying voice, dynamically allocates portions within a frame.

[Sakai, *et al.*, 1977]
T. Sakai, T. Hayashi, S. Kitazawa, K. Tabata, and T. Kadade, "Inhouse computer network Kulpnet", *Information Processing 77 (Proc. IFIP 77)*, Toronto, August 1977, pp. 161-166.
Star configuration with point-to-point lines (~1 Mbps) to a simple switching machine.

[Saltzer & Pogran, 1979]
J. H. Saltzer and K. T. Pogran, "A star-shaped ring network with high maintainability," *Local Area Computer Networks Symposium*, Boston, May 1979, pp. 179-190.

[Scantlebury, *et al.*, 1968]
R. A. Scantlebury, P. T. Wilkinson and K. A. Bartlett, "The design of a message switching centre for a digital communication network", *IFIP Congress 68*, Edinburgh, August 1968.

[Scantlebury, 1969]
R. A. Scantlebury, "A model for the local area of a data communication network -- objectives and hardware organization", *Proceedings of the ACM Symposium on Problems in the Optimization of Data Communication Systems [1st Data Communication Symposium]*, ACM SigComm, Pine Mountain, Georgia, October 1969, pp. 183-204.
Local system for NPL, mainly for terminals connected through multiplexers to a single central switch.

[Scantlebury & Wilkinson, 1971]
R. A. Scantlebury and P. T. Wilkinson, "The design of a switching system to allow remote access to computer services by other computers and terminal devices", *2nd Symposium on Problems in the Optimization of Data Communication Systems [2nd Data Communication Symposium]*, ACM SigComm and IEEE, Palo Alto, California, October 1971. Reprinted in [Chu, 1976].

[Scantlebury & Wilkinson, 1974]
R. A. Scantlebury and P. T. Wilkinson, "The National Physical Laboratory data communication network", *Second International Conference on Computer Communication (ICCC 74)*, Stockholm, August 1974, pp. 223-228.
Describes use of the NPL network, supporting about 75 terminals and 12 hosts -- connected through a single packet switch.

[Schenkel, 1974]
K. D. Schenkel, "An integrated 300 mbit/s time division multiplexed communication system with decentralized switches", *International Zurich Seminar on Digital Communication*, March 1974, paper D3.
Bus structure with no central control; two parallel send and receive trees. Proposal to use a branching tree utilizing fiber optics; actual system had only 1 branch, 5 hosts, used coaxial cable.

[Schultz & Davis, 1979]
J. R. Schultz and L. Davis, "The technology of PROMIS", *Proc. of the IEEE*, 67:9, September 1979, pp. 1237-1244.
Overview of the PROMIS medical records system; uses a two-way CATV communications system derived from work at Mitre.

[Schwartz, 1977]
Mischa Schwartz, *Computer-communication network design and analysis*, Prentice-Hall, 1977.
Esp chapter 12, "Polling in networks" and chapter 13, "Random access techniques."

[Sharma, *et al.*, 1974]
R. L. Sharma, J. C. Shah, M. T. El-Bardai, and K. K. Sharma, "C-system: multiprocessor network architecture", *Information Processing 74 (Proc. of IFIP 74)*, Stockholm, August 1974.
Uses a backbone TDM loop to connect major peripherals; a slower TDM loop to connect terminals.

[Sharma, *et al.*, 1979]
R. L. Sharma, A. D. Ingle, and P. T. de Sousa "Performance of some C-system configurations", *Computer Networking Symposium*, IEEE Computer Society/NBS, Gaithersburg, December 1979, pp. 26-35.
Performance results on the C-system multiprocessor with time division loops; uses a backbone loop among processors and a slower I/O loop for peripherals.

[Shatzer, 1978a]
Robert R. Shatzer, "Distributed Systems/1000", *Hewlett-Packard Journal*, March 1978, pp. 15-20.
Point to point lines, store-and-forward processing through the host, static routing tables (can be reset by hand if there is a failure).

[Shatzer, 1978b]
Robert R. Shatzer, "A minicomputer-based resource sharing datagram network", *Trends and Applications 1978: Distributed Processing*, IEEE and NBS Symposium, May 1978.
A nicely written paper, describing a full architecture from a datagram network up to user programs. Unfortunately, the "store-and-forward via host" technique introduces significant delays.

[Shatzer, *et al.*, 1979]
R. R. Shatzer, L. C. Hartge, A. P. Russon, and J. D. Chisholm, "HP's network concept stresses resource sharing and flexibility", *Data Communications*, August 1979, pp. 73-82.
Further discussion of HP's DSN architecture, and examples of the store-and-forward implementation in DS/1000.

[Sherman & Gable, 1977]
R. H. Sherman and M. Gable, "Microprocessor based networks applied to manufacturing control systems", *Proceedings of the National Electronics Conference*, 31, Chicago, October 1977, pp. 27-28.
Brief note on a manufacturing control application, used as an early test of an Ethernet-like system.

[Sherman, *et al.*, 1978a]
R. H. Sherman, M. Gable, and G. McClure, "Current summary of Ford activities in local networking", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 22-23.

[Sherman, *et al.*, 1978b]
R. H. Sherman, M. Gable, and G. McClure, "Concepts, strategies for local data network architectures", *Data Communications*, July 1978.

[Shoch, 1979a]
John F. Shoch, *Design and performance of local computer networks*, University Microfilms, August 1979.
Includes a general discussion and taxonomy of local networks, and detailed performance studies of the Ethernet system. A revised and expanded version of this work is to be published by McGraw-Hill in 1980.

[Shoch, 1979b]
John F. Shoch, *An annotated bibliography on local computer networks (1st edition)*, Xerox Parc Technical Report SSL-79-5, and IFIP Working Group 6.4 Working Paper 79-1, October 1979. Revised second edition published in February, 1980.

[Shoch & Hupp, 1979]
John F. Shoch and Jon A. Hupp, "Performance of an Ethernet local network -- a preliminary report", *Local Area Communications Network Symposium*, Boston, May 1979, pp. 113-125. Revised version published in *Proc. of 20th IEEE Computer Soc. Int. Conf. (Compcon '80 Spring)*, San Francisco, February 1980, pp. 318-322.

[Sideris, 1979]
George Sideris, "IEEE-488 standard spawns differing LSI interface designs", *Electronic Design*, June 7, 1979, pp. 23-24.

[Skatrud & Metz, 1976]
R. O. Skatrud and W. C. Metz, "Loop communications within supermarket store systems", *International Conference on Communications (ICC '76)*, Philadelphia, June 1976, pp. 30-6 - 30-11.

[Sloan, 1976]
Lansing J. Sloan, "A new design for interfacing computers to the Octopus network", *Conference on Experiments in New Approaches to Local Computer Networking*, U. of Minn., St. Paul, September 1976.

[Smith, 1975]
E. K. Smith, "Pilot two-way CATV systems", *IEEE Trans. on Comm.*, com-23:1, January 1975, pp. 111-120.
Reviews several proposals for two-way cable systems, and describes the Mitrix data communications system done at Mitre.

[Smith, 1979]
S. M. Smith, "Use of broadband coaxial cable networks in an assembly plant environment," *Proc. of the Local Area Communications Network Symposium*, Boston, May 1979, pp. 67-74.
Single-cable, two-way CATV system, used in 12 plants. Supports video, plus data (up to 48 Kbps). Equipment supplied by American Model Corp.

[Spaniol, 1979]
O. Spaniol, "Modelling of local computer networks," *Computer Networks*, 3:5, November 1979, pp. 315-326.
Analytic treatment of a proposed "Slotted Ethernet." Requires fixed-length packets, synchronized time base, etc.

[Sperry-Univac, 1977]
Sperry-Univac, *AN/USQ-67 converter-switching system, signal data*, Sperry-Univac Defense Systems, 1977.
64*(16/64) centralized switch, for inter-connecting peripherals, hosts, etc. Reports reduction of cable weight from 121 tons to 2.5 tons.

[Sperry-Univac, undated (1978?)]
Sperry-Univac, *AN/UYC 501(V) -- Shinpads system data bus* (product brochure), Sperry-Univac, undated (1978?).
Cable bus for shipboard use, with a central controller.

[Spragins, 1971]
J. D. Spragins, "Analysis of loop transmission systems", *Second Symposium on Problems in the Optimization of Data Communication Systems [2nd Data Communication Symposium]*, Palo Alto, October 1971, pp. 175-182.
Loop configurations: one CPU with multiple terminals, using centralized control.

[Spragins, 1972a]
J. D. Spragins, "Loops used for data collection", *Symposium on Computer-Communications Networks and Teletraffic, (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 59-76.
Model for only in-bound traffic on a loop, from terminals to a central controller.

[Spragins, 1972b]
J. D. Spragins, "Loops transmission systems -- mean value analysis", *IEEE Trans. on Comm.*, com-20:3, June 1972, pp. 592-602.
Loops with central control and fixed slots.

[Spragins, *et al.*, 1979]
J. Spragins, H. Jafari, and T. Lewis, "Some simplified performance modeling techniques with applications to a new ring-structured microcomputer network," *6th Annual Symposium on Computer Architecture*, Philadelphia, April 1979, pp. 111-116.

[Springer, 1978]
Joseph F. Springer, "The distributed data network, its architecture and operation", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, DC, September 1978, pp. 221-228.
A space division switch, "fast circuit switching."

[Starkson, 1979]
R. O Starkson, "A triaxial bus transmission system", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 82-85.
Transmission system for the Sperry Univac Shinpads (bus with central controller).

[Steward, 1970]
E. H. Steward, "A loop transmission system", *International Conference on Communications (ICC '70)*, San Francisco, June 1970, pp. 36-1 - 36-9.

[Switzer, 1972]
I. Switzer, "The cable television system as a computer-communications network", *Proc. of the Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972. Reprinted in [Green & Lucky, 1975].
Background on cable TV, possible ways to carry data.

[Szurkowski, 1978]
Edward Szurkowski, "A high bandwidth local computer network", *17th IEEE Computer Society International Conference (COMPCON Fall '78)*, Washington, September 1978, pp. 98-103.
A central PDP-11/70 with a string of micro-processors for data acquisition; a "super Unibus" with a single bus controller.

[Takahashi, *et al.*, 1979]
O. Takahaski, I. Fudemoto, A. Kusayanagi, and T. Akiba, "A new approach to multi-service in-house network (an optical intelligent highway)," *International Communications Conference (ICC '79)*, Boston, June 1979, paper 24.2.

Fiber optic loop at 6 Mbps; carries data and digitized voice.

[Teichholtz, 1975]
N. A. Teichholtz, "Digital network architecture", *European Computer Conference on Communication Networks (Online)*, London, September 1975, pp. 13-24.
A very early discussion of the broad outlines of DEC's DNA: DDCMP, NSP, DAP; little hard information.

[Thacker, *et al.*, 1979]
C. P Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto: A personal computer", Xerox Parc Technical Report, 1979. To appear in Siewiorek, Bell, and Newell, *Computer Structures: Readings and Examples, second edition*, 1979.
Describes the Alto computer, including its Ethernet interface (section 5).

[Thornton, 1979]
James E. Thornton, "Overview of Hyperchannel", *18th IEEE Computer Society International Conference (COMPCON Spring 79)*, San Francisco, February 1979, pp. 262-265.
A good clear discussion of the Hyperchannel.

[Thornton, 1980a]
James E. Thornton, "Some experience with high speed local computer networks", *Proc. of the 13th Hawaii Internation Conference on System Sciences*, Honolulu, January 1980, pp. 159-165.

[Thornton, 1980]
James E. Thornton, "Back-end network approaches", *Computer*, 13:2, February 1980, pp. 10-17.
General discussion of "back-end" file systems, with a detailed discussion of the NSC Hyperchannel.

[Thornton, *et al.*, 1975]
James E. Thornton, Gary S. Christensen, and Peter D. Jones, "A new approach to network storage management", *Computer Design*, November 1975, pp. 81-85.

[Three Rivers, 1979(?)]
Three Rivers Computer Corp., *PERQ, A landmark computer system*, undated product brochure (probably 1979).
Single user machines, tied together with a 10 Mbps cable system.

[Thurber, 1980]
Kenneth J. Thurber, "Perspective on local networks", *Proc. of the 13th Hawaii International Conference on System Sciences*, Honolulu, January 1980, pp. 155-158.
Revised version of an earlier article, [Thurber & Freeman, 1979a].

[Thurber & Freeman, 1979a]
Kenneth J. Thurber and Harvey A. Freeman, "Local computer network architectures", *18th IEEE Computer Society International Conference (COMPCON Spring '79)*, San Francisco, February-March 1979, pp. 258-261.

[Thurber & Freeman, 1979b]
Kenneth J. Thurber and Harvey A. Freeman, "A bibliography of local computer network architectures", *Computer Architecture News (ACM SigArch)*, 7:5, February 1979. Also published in *Computer Communication Review (ACM SigComm)*, 9:2, April 1979.
Revised version of their earlier article, [Thurber & Freeman, 1979a].

[Thurber & Freeman, 1979c]
Kenneth J. Thurber and Harvey A. Freeman, "Architectural considerations for local computer networks", *1st Int. Conf. on Distributed Computing Systems*, Huntsville, October 1979, pp. 131-142.

[Tobagi & Hunt, 1979]
F. A. Tobagi and V. B. Hunt, "Performance analysis of carrier sense multiple access with collision detection", *Local Area Communications Network Symposium*, Mitre and NBS Boston, May 1979, pp. 217-245. Also issued as Technical Report 173, Computer Systems Laboratory, Stanford University, June 1979.

[Tokoro & Tamaru, 1977]
M. Tokoro and K. Tamaru, "Acknowledging Ethernet", *15th IEEE Computer Society International Conference (COMPCON Fall '77)*, Washington, September 1977, pp. 320-325.

[Tominaga, *et al.*, 1976]
H. Tominaga, M. Kosuga, Y. Taniwaki, and K. Arata, "On a loop switching network system selecting direction to transmission," *International Switching Symposium*, Kyoto, October 1976, paper 133-3.
Analytic treatment of a "bi-directional" ring.

[Tropper, 1979]
Carl Tropper, *Models of local computer networks*, MTR-3783, Mitre Corp., May 1979.
Review and comparision of previous work on analytic and simulation models of local networks.

[Tsuchiya, *et al.*, 1974]
M. Tsuchiya, S. S. Yau, and M. J. Gonzalez, "Use of a computer network as peripheral devices", *8th IEEE Computer Society International Conference (COMPCON '74)*, San Francisco, February 1974, pp. 117-119.
Simple star configuration.

[Van den Bos, 1977]
J. Van den Bos, "A design of a communication supervisor for a local network employing monitors", *International Computing Symposium*, ACM, Liege, Belgium, April 1977, pp. 319-323.
A simple star system, giving multiple PDP-11's access to a central 370.

[Venetsanopoulos & Newhall, 1970]
A. N. Venetsanopoulos and E. E. Newhall, "Traffic considerations in an experimental distributed switching system", *Canadian Symposium on Communications*, IEEE, November 1970, p. 37.
Abstract only, no paper.

[Wanner, 1978]
James F. Wanner, "Wideband communication system improves response time", *Computer Design*, December 1978, pp. 85-92.
CATV bus, uses central controller with polling of other terminals.

[Watson, 1978]
Richard W. Watson, "The LLL Octopus network: some lessons and future directions", *3rd USA-Japan Computer Conference*, San Francisco, October 1978, pp. 12-21.
A good review of the development of the Octopus system, and some of the problems encountered.

[Watson, 1979]
W. Bruce Watson, "Simulation study of the traffic dependent performance of a prioritized, CSMA broadcast network", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 67-74.
Further simulation work on the NSC Hyperchannel.

[Wecker, 1974]
S. Wecker, "Dialog: Advanced link control runs full and half duplex on various types of nets", *Data Communications*, September/October 1974, pp. 36-46.
Describes DDCMP, Digital's line control procedure.

[Wecker, 1975]
S. Wecker, "Interchange: Packet switching with assorted computers in a private network", *Data Communications*, March/April 1975, pp. 56-63.
Describes DDCMP, Digital's line control procedure.

[Wecker, 1976a]
Stuart Wecker, "The design of DECNET -- a general purpose network base", *IEEE Electro '76*, Boston, May 1976.

[Wecker, 1976b]
Stuart Wecker, "DECNET: a building block approach to network design", *National Telecommunications Conference (NTC '76)*, Dallas, November 1976, paper 7.5.

[Wecker, 1978]
Stuart Wecker, "DECNET: Issues related to local networking", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 26-31.

[Weller, 1971]
David R. Weller, "A loop communication system for I/O to a small multi-user computer", *5th Annual IEEE Computer Society International Conference*, Boston, September 1971, p. 77-78.
At Bell Labs loop for I/O to a small host, fixed size 35 bits/frame.

[West, 1972a]
L. P. West, "Loop-transmission control structures", *IEEE Trans. on Comm.*, com-20, June 1972, pp. 531-539. Reprinted in [Green & Lucky, 1975].

[West, 1972b]
L. P. West, "High available loop carrier system", *IBM Technical Disclosure Bulletin*, 15:1, June 1972, pp. 337-339.
A method to locate broken links on a terminal loop.

[West, 1977]
Anthony R. West, *A broadcast packet-switched computer communication network: design progress report*, TR 121, Computer System Laboratory, Queen Mary College, London, February 1977.

[West, 1978]
Anthony R. West, "Local area networks at Queen Mary College", *Local Area Networking -- Report of a Workshop Held at the National Bureau of Standards, Aug. 22-23, 1977*, NBS Special Publication 500-31, 1978, pp. 23-26.

[West & Davison, 1978]
Anthony West and Allan Davison, *CNET -- A cheap network for distributed computing*, TR 120, Computer System Laboratory, Queen Mary College, March 1978.

[White & Maxemchuk, 1974]
H. E. White and N. F. Maxemchuk, "An experimental TDM data loop exchange", *International Conference on Communications (ICC '74)*, Minneapolis, June 1974, paper 7A.
TDM loop, fixed channels up to 9600 bps, run by a loop clock. Several loops, connected through a switch.

[Wilkes, 1975]
M. V. Wilkes, "Communication using a digital ring", *PACNET Conference*, Sendai, Japan, August 1975. (See also [Wilkes & Wheeler, 1976].)

[Wilkes & Needham, 1980]
M. V. Wilkes and R. M. Needham, "The Cambridge model distributed system," *ACM Operating Systems Review*, 14:1, January 1980, pp. 21-29.
Proposed plans for using the Cambridge ring.

[Wilkes & Wheeler, 1976]
M. V. Wilkes and D. J. Wheeler, "Design of a digital ring -- Addendum to a a paper presented to the PACNET Conference held in Sendai Japan, in August 1975", December 1976.

[Wilkes & Wheeler, 1979]
M. V. Wilkes and D. J. Wheeler, "The Cambridge digital communication ring", *Proc. of the Local Area Communications Network Symposium*, Mitre and NBS, Boston, May 1979, pp. 47-61.

[Wilkinson, 1969]
P. T. Wilkinson, "A model for the local area of a data communication network -- software organization", *Proceedings of the ACM Symposium on Problems in the Optimization of Data Communication Systems [1st Data Communication Symposium]*, ACM SigComm, Pine Mountain, Georgia, October 1969, pp. 155-181.
NPL system, basically terminals connected to a star.

[Wilkinson & Scantlebury, 1968]
P. T. Wilkinson and R. A. Scantlebury, "The control functions in a local data network", *IFIP Congress 68*, Edinburgh, August 1968.
Describes the single Interface Computer (IC) in each local area, connected to the S/F backbone.

[Will, 1970]
Craig Will, "The data ring: a communication facility for the DCS", *Appendix 2, Supplement to proposal for research submitted to the National Science Foundation on Distributed Computing System*, University of California, Irvine, October 1970.
Very early material; includes a comparison of control passing vs. empty slot.

[Willard, 1973]
David G. Willard, "MITRIX: a sophisticated digital cable communications system", *National Telecommunications Conference (NTC '73)*, Atlanta, November 1973, paper 38E.

[Willard, 1974]
David G. Willard, "A time division multiple access system for digital communication", *Computer Design*, June 1974, pp. 79-83.

[Williams, 1979]
Ronald M. Williams, "LSI chips ease standard 488 bus interfacing", *Computer Design*, October 1979, pp. 123-131.
Specific discussion of the Intel chips; for a brief comparison of LSI implementations of the 488 standard, see [Sideris, 1979].

[Wolf & Liu, 1978]
J. J. Wolf and M. T. Liu, "A distributed double-loop computer network (DDLCN)," *7th Texas Conference on Computing Systems*, November 1978, pp. 6-19 - 6-34.

[Wolf, *et al.*, 1979a]
J. J. Wolf, M. T. Liu, B. W. Weide, and D. P. Tsay, "Design of a distributed fault-tolerant loop network," *9th Annual International Symposium on Fault-tolerant computing (FTCS-9)*, Madison, June 1979, pp. 17-24.

[Wolf, *et al.*, 1979b]
J. J. Wolf, B. W. Weide, and M. T. Liu, "Analysis and simulation of the distributed double-loop computer network," *Computer Networking Symposium*, IEEE(CS)/NBS, Gaithersburg, December 1979, pp. 82-89.

[Wong, *et al.*, 1978]
J. W. Wong, J. A. Field, and S. N. Kalra, "Feasibility of a loop system for local data concentration", *International Conference on Communications (ICC '78)*, Toronto, June 1978.
Reserved model of a low-speed loop used to connect terminals to a concentrator, for access to a host. Reliability estimates for 3 different schemes to bypass broken components, and some simple discrete simulations.

[Wood, *et al.*, 1979]
D. C. Wood, S. F. Holmgren, A. P. Skelton, "A cable-based protocol architecture," *6th Data Comm. Sym.*, Pacific Grove, November 1979, pp. 137-146.
The Cablenet project, using one of the Mitre CSMA/LWT CATV systems.

[Wu & Chen, 1975]
R. M. Wu and Y. Chen, "Analysis of a loop transmission system with round-robin scheduling of services", *IBM Journal of Research and Development*, September 1975, pp. 486-493.

[Yajima, *et al.*, 1977a]
Shuzo Yajima, Yahiko Kambayashi, Susumu Yoshida, and Kazuo Iwama, "Optically linked laboratory computer network Labolink", *Proceedings of the 10th Hawaii International Conference on System Sciences*, Honolulu, January 1977, pp. 1-4.
Simple star topology, but uses fiber optic links.

[Yajima, *et al.*, 1977b]
S. Yajima, Y. Kambayashi, S. Yoshida, and K. Iwama, "Labolink: an optically linked laboratory computer network", *Computer*, IEEE Computer Society, November 1977, pp. 52-59.
Simple star topology, but uses fiber optic links.

[Yao, 1978]
A. C. Yao, "On the loop switching addressing problem", *SIAM Journal of Computing*, 7:4, November 1978, pp. 515-523.

[Yasaki, 1978]
E. K. Yasaki, "IBM's offering of SNA: some find it a success", *Datamation*, February 1978, pp. 176-177.
Reports on results of a survey of 15 large SNA installations.

[Yatsuboshi, *et al.*, 1978]
R. Yatsuboshi, T. Tsuda, K. Yamaguchi and Y. Inoue, "An in-house network configuration for distributed intelligence", *Fourth International Conference on Computer Communication (ICCC 78)*, Kyoto, September 1978.
Hybrid hierarchy of loops: 48 Kbps HDLC loops for terminals, 6.3 Mbps ring as a Data Highway, or backbone.

[Yeh, 1979]
Jeffry W. Yeh, "Simulations of local comuter networks", *4th conference on local computer networks*, Minneapolis, October 1979, pp. 56-66.
Reviews methodology and results from Hyperchannel simulations.

[Yu & Majithia, 1979]
L. W. Yu and J. C. Majithia, "An adaptive loop-type data network", *Computer Networks*, 3:2, April 1979, pp. 95-104.
Proposal for a full-duplex empty slot (Pierce) ring, with two slot sizes and an "adaptive priority scheme." Small packets for control; used to set priorities from a loop supervisor.

[Yuen, *et al.*, 1972]
M. L. T. Yuen, B. A. Black, E. E. Newhall, and A. N. Venetsanopoulos, "Traffic flow in a distributed loop switching system", *Symposium on Computer-Communications Networks and Teletraffic (Polytechnic Institute of Brooklyn, April 1972)*, Polytechnic Press, 1972, pp. 29-46. Reprinted in [Chu, 1976].
Simulation of the ring at 403 kbits/sec and 3.2 megabits/sec. Each unit on the loop is a Teletype; requires an 8-bit buffer in each.

[Zafiropulo, 1973]
P. Zafiropulo, "Reliability optimization in multiloop communication networks", *IEEE Trans. on Comm*, com-21:8, August 1973, pp. 898-907.
Analytical derivation for average availability in a hierarchy of loops. Shows that even with very large numbers of terminals, more than 3 stages adds little to the reliability.

[Zafiropulo, 1974a]
P. Zafiropulo, "Reliability -- a key element in loop systems", *International Zurich Seminar on Digital Communication*, March 1974, paper D2.
Assessment of techniques for using a stand-by loop, with bypass and self-heal actions; procedures for reconfiguration.

[Zafiropulo, 1974b]
P. Zafiropulo, "Performance evaluation of reliability improvement techniques for single-loop communications systems", *IEEE Trans. on Comm.*, com-22:6, June 1974.
Expanded version of [Zafiropulo, 1974a].

[Zafiropulo & Rothauser, 1972]
P. Zafiropulo and E. H. Rothauser, "Signalling and frame structures in highly decentralized loop systems", *First International Conference on Computer Communication (ICCC '72)*, Washington, October 1972, pp. 309-315.
Proposal for a loop in which a loop controller circulates an empty frame structure. Partitioned into 2 parts, one for regular speech traffic and one for data.

ANNEXE B

Projet de Stage

Université de Lille

LE MODE DE BASE

1. Les caractéristiques fonctionnelles du mode de base

- Le mode de base s'emploie sur des liaisons point à point ou multipoint centralisé;
- Le code utilisé est l'alphabet international no. 5 (CCITT no.5 ou ASCII);
- La transmission peut théoriquement être synchrone ou asynchrone mais en pratique la procédure utilise la transmission synchrone;
- Le mode d'exploitation est bilatéral à l'alternat.

2. Code utilisé

L'alphabet internationnal no. 5 fournit 128 combinaisons de code, ou caractères, dont vingt sont réservées au contrôle de liaison. Pour chaque caractère on peut utiliser un élément de parité qui doit être impair pour la transmission synchrone et pair pour la transmission asynchrone. La transmission s'effectue avec les bits de poids le plus faible en tête.

Les dix caractères utilisés pour le contrôle sont les suivants (Annexe 2) :

SOH (Start of header) DEBUT D'EN-TETE
Caractère de commande de transmission employé pour indiquer le commencement d'un en-tête de message d'information.

STX (Start of text) DEBUT DE TEXTE
Caractère de commande de transmission précédant en texte et employé pour terminer un en-tête.

ETX (End of text) FIN DE TEXTE
Caractère de commande de transmission utilisé pour terminer un texte.

EOT (End of transmission) FIN DE TRANSMISSION
Caractère de transmission utilisé pour indiquer la fin de la transmission d'un ou de plusieurs textes.

ENQ (Enquiry) DEMANDE
Caractère de commande de transmission employé comme demande de réponse d'une station éloignée -la réponse peut inclure l'identification et l'état de la station . Lorsqu'un contrôle d'identité ("Qui est là ?") est exigé sur un réseau général de transmission avec communtation, la première utilisation du caractère ENQ après l'établissement de la liaison a le sens de la question "Qui est là ?". Une nouvelle utilisation de ce caractère peut ou non inclure la fonction "Qui est là ?" selon accord préalable.

ACK (Acknowledge) ACCUSE DE RECEPTION
Caractère de commande de transmission transmis par un récepteur comme

réponse affirmative à l'émetteur.

DLE (Data link escape) ECHAPPEMENT DE TRANSMISSION
Caractère de commande qui change la signification d'un nombre limité de caractères successifs qui le suivent. Ce caractère est utilisé exclusivement pour fournir des commandes supplémentaires de transmission.

NAK (Negative acknowledge) ACCUSE DE RECEPTION NEGATIF
Caractère de commande de transmission transmis par un récepteur comme réponse négative à l'émetteur.

SYN (Synchronous idle) SYNCHRONISATION
Caractère de commande de transmission utilisé par un système de transmission synchrone en l'absence de tout autre caractère (situation inactive) pour produire un signal à partir duquel le synchronisme peut être obtenu ou maintenu entre des équipements terminaux de données.

ETB (End of transmission block) FIN DE BLOCK DE TRANSMISSION
Caractère de commande de transmission utilisé pour indiquer la fin d'un bloc de données, lorque ces données sont divisées en blocs en vue de leur transmission.

3. Etat des stations

Il est nécessaire de préciser les états de chaque station de la liaison afin de limiter sa responsabilité dans le fonctionnement normal comme dans les situations anormales.

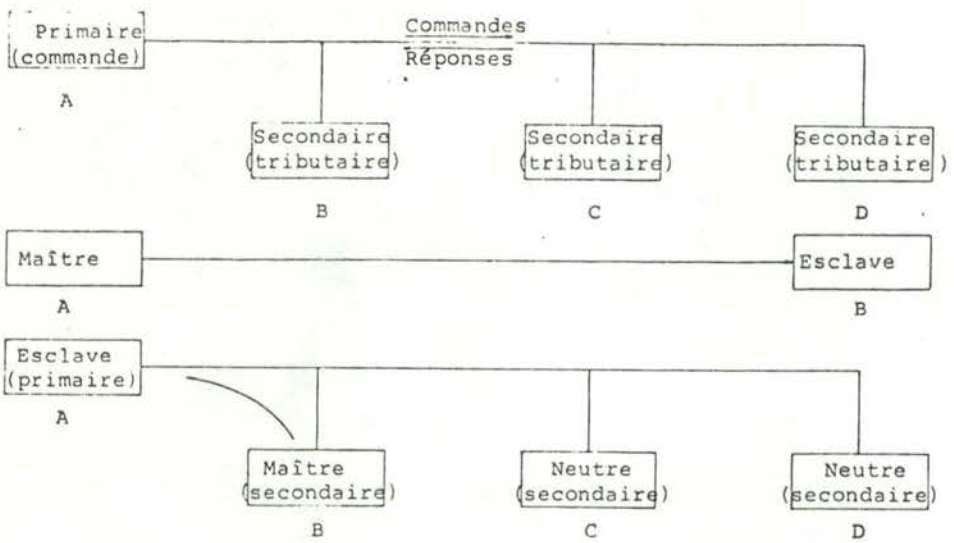


Fig. 1 Etats des stations.

Une station de données peut avoir des états permanents et des états temporaires.

Etats permanents

Dans une configuration en multipoint, comme en boucle, la supervision de la liaison et le contrôle de son fonctionnement sont en général attribués à une seule station. On dit que cette station est dans l'état primaire ou de commande. Les autres stations sont dans l'état secondaire ou tributaire (Fig. 1a). La fonction primaire consiste à envoyer des commandes vers la ou les secondaires. Elle interprète les réponses reçues des secondaires et d'une manière générale, doit :

- organiser l'échange des données;
- assurer la supervision de la liaison;
- assurer la responsabilité de reprise au niveau de la liaison, en cas de perturbation du fonctionnement.

La fonction secondaire consiste à exécuter les commandes envoyées par le primaire et, en conséquence, lui envoyer des réponses.

Etats temporaires

Outre les fonctions primaires, ou de commande, et secondaires, ou tributaires, qui sont assignées comme états permanents aux stations de données d'une liaison, on utilise aussi des états assignés de manière temporaire aux stations, selon que celles-ci sont, à un instant donné, émettrices ou réceptrices de texte, ou encore ni l'une ni l'autre. Ainsi dans la figure 1b, le texte, à l'instant considéré, étant émis par A et reçu par B, la station A est dite maître et la station B, esclave. Ces états ne durent que le temps de l'écoulement du texte et peuvent éventuellement changer après.
Dans la figure 1c , le texte est émis, à l'instant considéré, par la station secondaire B et reçu par la station primaire. B est alors la station maître et la station primaire est esclave. Les stations C et D qui, à cet instant, ne sont ni émettrices ni réceptrices, sont dans l'état temporaire neutre.

4. Format des blocs

Les blocs transmis par une liaison en mode de base sont de deux types : blocs d'informations et séquences de supervision.

4.1. Blocs d'information

Les procédures en mode de base permettent d'offrir à leurs usagers le transport de messages de longueur à priori quelconque. Ces messages peuvent être précédés d'un en-tête. La norme ne spécifie pas la nature de cet en-tête : certains l'utilisent pour des fonctions propres à la procédure de liaison (numérotage des blocs par exemple),

d'autres pour des fonctions d'un niveau supérieur étranger à la gestion de la liaison (indication de la nature du message par exemple).
On utilise les caractères de commande SOH, STX, ETX et ETB pour encadrer les blocs d'information. Considérons un message se composant d'un en-tête et d'un texte :

EN-TETE TEXTE

On commence par insérer les caractères de commande délimitant ce message, qui devient :

SOH EN-TETE STX TEXTE ETX

ou, s'il n'y a pas d'en-tête :

STX TEXTE ETX

Si le message est trop long, il peut être découpé en blocs de longueurs appropriées aux possibilités de trnasmission de la liaison. Dans ce cas, chaque bloc est alors complété par l'adjonction du caractère ETB, sauf le dernier qui doit être terminé par ETX, et précédé par le caractère STX ou SOH suivant qu'il s'agit d'une partie de l'en-tête ou du texte. Notre message peut devenir, par exemple :

Bloc 1 :	SOH	EN-TETE (1)	ETB		
Bloc 2 :	SOH	EN-TETE (2)	ETB	TEXTE (début)	ETB
Bloc 3 :	SOH	TEXTE (fin)	ETX		

Contrôle d'erreurs

La protection contre les erreurs de transmission se fait,

- soit par l'utilisation des parités longitudinale et transversales où le caractère de contrôle est appelé BCC (Block Check Character),

- soit par l'utilisation de code cyclique où le polynome générateur est celui de l'avis V41 du CCITT, c'est-à-dire :

$$X^{16} + X^{12} + X^{5}$$

4.2. Séquences de supervision

La procédure en mode de base Réalise les fonctions de supervision de la liaison (Acquittement, invitation à émettre, invitation à recevoir, etc.) en utilisant les caractères de commande EOT, ENQ, ACK et NAK. Ces caractères sont utilisés en conjonction avec des séquences d'adresse ou de préfixes. Les règles précises pour constituer ces séquences ne sont pas spécifiées. Ces séquences peuvent, par exemple, comprendre des informations relatives à l'adresse, à l'identité, à l'état de la station.
Le mode d'exploitation de la procédure étant bilatéral à l'alternat, le transfert de l'information doit s'effectuer sous contrôle, soit par des commandes d'invitation à émettre et à recevoir eznvoyées par une station de commande (Configuration multipoint), soit par une demande de recevoir envoyée par une station (Configuration point à point).
Outre la supervision du transfert de l'information, quelques autres commandes permettent de contrôler l'état de la liaison. Les principales fonctions de supervision sont donc :

Invitation à émettre

Pour inviter une station d'adresse A à émettre, la station de commande émet la séquence

EOT A ENQ

(Le caractère EOT sert à mettre ou à remettre la liaison dans l'état de commande).

- Si la station A n'a pas de message à transmettre, elle doit retourner la réponse EOT;
- Si elle a un bloc d'information à envoyer, elle l'envoie en utilisant le format des blocs d'information (SOH ...)
- Si elle ne répond pas au bout d'un temps donné, ou si la réponse est erronée ou perdue, la station de commande entre en procédure de reprise.

Invitation à recevoir

Pour inviter une station d'adresse B à recevoir, la station de commande émet la séquence

B ENQ

- En recevant une invitation à recevoir, si la station B est prête à recevoir, elle doit retourner à la station de commande la réponse ACK;
- Si la station B ne peut pas recevoir, elle doit répondre NAK;
- Si elle ne répond pas, ou si la réponse est erronée ou perdue, la station de commande entre en procédure de reprise.

Demande de réception

L'orsqu'il n'y a que deux stations sur la liaison, c'est-à-dire en configuration point à point, pour éviter un état de commande permanent par l'une des stations, la procédure permet à la liaison de fonctionner en mode contention. Dans ce mode, la station qui veut émettre envoie la demande de réception ENQ à la station opposée. Celle-ci doit répondre comme dans le cas d'une invitation à recevoir. En cas de contention, c'est-à-dire si les deux stations essayent de transmettre simultanément, on résout le conflit en donnant des valeurs différentes aux temporisateurs pour la reprise dans les deux stations.

Demande de réponse

Pour obtenir une réponse, ou la répétition d'une réponse à une commande envoyée, une station maître utilise le caractère ENQ.

Fin de transmission

L'envoi du caractère EOT par une station termine la transmission. Pour les liaisons comprenant une station de commande, cela signifie que la responsabilité de la commande de liaison est retournée à la station de commande. Pour les liaisons point à point sans stations de commande, l'envoi de EOT renvoie la liaison à l'état neutre.

Libération de la liaison

La séquence DLE EOT est utilisée pour libérer une liaison.

Synchronisation de caractères

En mode de base, on utilise le caractère SYN pour obtenir ou maintenir la synchronisation de caractères. A chaque transmission, on fait précéder la séquence à transmettre par des caractères SYN et pendant la transmission, on peut en insérer pour maintenir la synchronisation.

Procédures de reprise

Les procédures de reprise sont basées sur l'utilisation de temporisateurs et de compteurs. Lorsqu'une station de commande ou une station maître n'obtient pas de réponse valide à une séquence de supervision ou à un bloc d'information envoyé, à l'expiration du temps donné, elle peut :

- répéter la séquence ou le bloc, jusqu'à n fois;
- envoyer une demande de réponse (ENQ, n fois);
- terminer la transmission en envoyant EOT.

Au bout de n répétitions ou demandes de réponses sans succès, la station doit ,par exemple, informer le niveau supérieur du

transmis.

Mode conversationnel

Le mode conversationnel, qui est une extension de la procédure en

de transmission dû aux acquittements des blocs d'information;
- l'emploi d'un autre polynôme pour le contrôle d'erreurs par code cyclique.

Citons, enfin, la famille des procédures BSC (Binary Synchronous Communications) d'IBM qui est aussi une application du mode de base.

Annexe B2

Programmes du logiciel de la procédure TMM-RB

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE IDLEMOD
OBJECT MODULE PLACED IN IDLE.OBJ
COMPILER INVOKED BY:  PLM80 IDLE.PLM PRINT(:LP:) PAGEWIDTH(132) XREF DEBUG TITLE('IDLE') DATE(04.01.82) WORKFILES(:F0:,:F0:)


   1          IDLE$MOD: DO;
   2    1      IDLE: PROCEDURE PUBLIC;
   3    2      DECLARE FOREVER LITERALLY 'WHILE 1=1';
   4    2      DECLARE BASE ADDRESS;
   5    2       BASE=STACK$PTR;
   6    2       DO WHILE STACK$PTR=BASE;
   7    3       END;
   8    2      DISABLE; HALT;
  10    2     END IDLE;
  11    1     END IDLE$MOD;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE USER
OBJECT MODULE PLACED IN USER.OBJ
COMPILER INVOKED BY:  PLM80 USER.PLM PRINT(:LP:) PAGEWIDTH(132) XREF DEBUG TITLE('USER') DATE(04.01.82) WORKFILES(:F0:,:F0:)


   1          USER: DO;

                 $INCLUDE(:F0:TMSTR.ELT)
                =
   2    1      =  DECLARE     def$mres$syn    LITERALLY
               =  'STRUCTURE( link            ADDRESS,
               =              length          ADDRESS,
               =              type            BYTE,
               =              k$free$rec      ADDRESS,
               =              k$treat$rec     ADDRESS,
               =              k$prior$send    ADDRESS,
               =              k$norm$send     ADDRESS)';
                =
   3    1      =  DECLARE     def$bal         LITERALLY
               =  '           ptart           ADDRESS,
               =              kplein          ADDRESS,
               =              kvide           ADDRESS,
               =              lmin            ADDRESS';
                =
   4    1      =  DECLARE     def$hd$art$bal  LITERALLY
               =  '           link            ADDRESS,
               =              length          ADDRESS,
               =              type            BYTE,
               =              long            BYTE';
                =
   5    1      =  DECLARE     def$msg$ress    LITERALLY
               =  'STRUCTURE( msg$hdr,
               =              tres            BYTE)';
                =
   6    1      =  DECLARE     def$msg$bal     LITERALLY
               =  'STRUCTURE( def$hd$art$bal,
               =              tam(0)          BYTE)';
                =
   7    1      =  DECLARE     def$hdr$bloc    LITERALLY
               =  '           link            ADDRESS,
               =              length          ADDRESS,
               =              type            BYTE,
               =              home$exchange   ADDRESS,
               =              long            BYTE';
                =
   8    1      =  DECLARE     def$msg$bloc$rec LITERALLY
               =  'STRUCTURE( def$hdr$bloc,
               =              block (252)     BYTE)';
```

```
                =   $EJECT
      9   1     =   DECLARE     lart$cons       LITERALLY '72',
                =               lbal$cons       LITERALLY '1024',
                =               lart$impri      LITERALLY '133',
                =               lbal$impri      LITERALLY '512',
                =               lart$perfo      LITERALLY '80',
                =               lbal$perfo      LITERALLY '512';
                =
     10   1     =   DECLARE     t$cour$bal      LITERALLY '0',
                =               t$last$bal      LITERALLY '0',
                =               f$typ           LITERALLY '0',
                =               typ$b$libre     LITERALLY '0';  /*à définir*/
                =
     11   1     =   DECLARE     KC2$CONS        LITERALLY '62H',
                =               KC3             LITERALLY '20H',
                =               KETX            LITERALLY '03H',
                =               KETB            LITERALLY '17H';
```

```
                =   $EJECT
                =
                =   /* Messages types */
                =
     12    1    =   DECLARE typ$int             LITERALLY ' 1',     /* interrupt    */
                =           typ$trig$syn        LITERALLY '63',     /* Triggering   */
                =           typ$free$syn        LITERALLY '64',     /* free         */
                =           typ$alarm$syn       LITERALLY '65',     /* alarm        */
                =           typ$wrong$syn       LITERALLY '66',     /* wrong        */
                =           typ$err$blocked     LITERALLY '67',     /* error+blocked*/
                =           typ$stop$syn        LITERALLY '68',     /* stop request */
                =           typ$res$syn         LITERALLY '69',     /* restart req. */
                =           typ$rec$keyboard    LITERALLY '70',     /* keyboard req.*/
                =           typ$rec$print$ETB   LITERALLY '71',     /* printer req. */
                =           typ$rec$card$ETB    LITERALLY '72',     /* card reader  */
                =           typ$over$timed      LITERALLY '03',     /* time out RMX */
                =           typ$res$res         LITERALLY '73';     /* res. reserva.*/
                =
```

```
                $EJECT

                /* EXTERNAL PROCEDURES */

 13   1         EXIT:
                    PROCEDURE EXTERNAL;
 14   2          END EXIT;

 15   1         ERROR:
                    PROCEDURE (STATUS) EXTERNAL;
 16   2          DECLARE STATUS ADDRESS;
 17   2          END ERROR;

 18   1         READ:
                    PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
 19   2          DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
 20   2          END READ;

 21   1         RQSEND:
                    PROCEDURE (exchange$address,message$address) EXTERNAL;
 22   2          DECLARE (exchange$address,message$address) ADDRESS;
 23   2          END RQSEND;

 24   1         RQWAIT:
                    PROCEDURE (exchange$address,time$limit) ADDRESS EXTERNAL;
 25   2          DECLARE (exchange$address,time$limit) ADDRESS;
 26   2          END RQWAIT;

 27   1         DECLARE (K$free$rec,
                         K$treat$rec,
                         K$norm$send,
                         K$res$syn,
                         inlex0,
                         K$prior$send)  ADDRESS EXTERNAL;
 28   1         DECLARE K$home$user     ADDRESS EXTERNAL;   /* User's home$exchange*/
 29   1         DECLARE OUT$EN          ADDRESS PUBLIC;     /* 0=OUTPUT ENABLE */
                                                            /* 1=OUTPUT DISABLE*/
```

```
                $EJECT

 30   1         DECLARE KEYBOARD          LITERALLY '1';
 31   1         DECLARE (i,j)             BYTE;
 32   1         DECLARE dummy             ADDRESS;
 33   1         DECLARE msg$base          ADDRESS;
 34   1         DECLARE MESSAGE$STRUCTURE LITERALLY
                 'STRUCTURE  (link            ADDRESS,
                             length          ADDRESS,
                             type            BYTE,
                             home$exchange   ADDRESS,
                             block$length    BYTE,
                             block(252)      BYTE)';

 35   1         DECLARE mress BASED msg$base def$mres$syn;
 36   1         DECLARE message MESSAGE$STRUCTURE;
```

```
                $EJECT

 37   1     USER:
                PROCEDURE PUBLIC;
 38   2          DECLARE (actual,status) ADDRESS;
 39   2          msg$base = RQWAIT (.K$res$syn,0);    /* Gerant available ? */
 40   2          mress.type =63;                      /* typ$trig$syn       */
 41   2          mress.k$free$rec  =.k$free$rec;
 42   2          mress.k$treat$rec =.k$treat$rec;
 43   2          mress.k$prior$send=.k$prior$send;
 44   2          mress.k$norm$send =.k$norm$send;
 45   2          CALL RQSEND (.inlex0,msg$base); /* Trigger the task gerant */

 46   2     CONT:
             CALL READ (KEYBOARD,.message.block(2),128,.actual,.status);
 47   2          IF status()0
              THEN DO;
 49   3               CALL ERROR (status);
 50   3               CALL EXIT;
 51   3               END ;
 52   2          IF message.block(2)='('
              THEN DO;
 54   3                CALL RQSEND(.OUT$EN,.message) /* .message = dummy msg */;
 55   3                GOTO CONT;
 56   3                END;
 57   2          message.block(0)         = 62H;          /* C2                */
 58   2          message.block(1)         = 20H;          /* C3                */
 59   2          message.type             = 72;           /* message type      */
 60   2          message.block(actual)    = 17H;          /* message (ETB)     */
 61   2          message.block$length     = actual+1;     /* message length    */
 62   2          message.home$exchange    = .K$home$user; /* Ack by K$free$rec*/
 63   2          CALL RQSEND (.K$prior$send,.message);
 64   2          DO WHILE RQWAIT(.K$home$user,0)().message;
 65   3               END;
 66   2          GOTO CONT;
 67   2          END USER;
 68   1     END USER;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SCREEN
OBJECT MODULE PLACED IN SCREEN.OBJ
COMPILER INVOKED BY:  PLM80 SCREEN.PLM PRINT(:LP:) PAGEWIDTH(132) XREF DEBUG TITLE('SCREEN') DATE(04.01.82) WORKFILES(:F0:,:F0:)


  1          SCREEN: DO;

  2   1      DECLARE CR      LITERALLY '0DH',
                     LF      LITERALLY '0AH';

  3   1      DECLARE TRUE    LITERALLY '0FFH',
                     FOREVER LITERALLY 'WHILE TRUE';

                 /*  EXTERNAL PROCEDURES */

  4   1      CO:
                 PROCEDURE(CHAR) EXTERNAL;
  5   2          DECLARE CHAR BYTE;
  6   2          END CO;

  7   1      RQWAIT:
                 PROCEDURE (exchange$address,time$limit) ADDRESS EXTERNAL;
  8   2          DECLARE (exchange$address,time$limit) ADDRESS;
  9   2          END RQWAIT;

 10   1      RQSEND:
                 PROCEDURE (exchange$address,message$address) EXTERNAL;
 11   2          DECLARE (exchange$address,message$address) ADDRESS;
 12   2          END RQSEND;

 13   1      DECLARE (K$free$rec,K$treat$rec) ADDRESS EXTERNAL;
```

```
                $EJECT

  14   1        SCREEN:
                    PROCEDURE PUBLIC;

  15   2            DECLARE msg$base     ADDRESS;        /* dummy : not used */
  16   2            DECLARE message
                            STRUCTURE    (link           ADDRESS,
                                          length         ADDRESS,
                                          type           BYTE,
                                          home$exchange  ADDRESS,
                                          block$length   BYTE,
                                          block(252)     BYTE);
  17   2            DECLARE status       ADDRESS,
                            i            BYTE;

  18   2            DO FOREVER;
  19   3                CALL RQSEND(.K$free$rec,.message);
  20   3                msg$base=RQWAIT(.K$treat$rec,0);
  21   3                CALL CO(LF);
  22   3                CALL CO(CR);
  23   3                DO i=1 TO message.block$length;
  24   4                    CALL CO(message.block(i));
  25   4                    END;
  26   3                END;
  27   2            END SCREEN;
  28   1        END SCREEN;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE RESULT
OBJECT MODULE PLACED IN RESULT.OBJ
COMPILER INVOKED BY:  PLM80 RESULT.PLM PRINT(:LP:) PAGEWIDTH(132) XREF DEBUG TITLE('RESULT') DATE(04.01.82) WORKFILES(:F0:,:F0:)


   1          RESULT: DO;

                      /* EXTERNAL PROCEDURES */

   2   1      EXIT:   PROCEDURE EXTERNAL;
   3   2              END EXIT;
   4   1      ERROR:  PROCEDURE (STATUS) EXTERNAL;
   5   2                  DECLARE STATUS ADDRESS;
   6   2              END ERROR;
   7   1      WRITE:  PROCEDURE (AFTN,BUFFER,COUNT,STATUS) EXTERNAL;
   8   2                  DECLARE (AFTN,BUFFER,COUNT,STATUS) ADDRESS;
   9   2              END WRITE;
  10   1      OPEN:   PROCEDURE (AFTN,FILE,ACCESS,MODE,STATUS) EXTERNAL;
  11   2                  DECLARE (AFTN,FILE,ACCESS,MODE,STATUS) ADDRESS;
  12   2              END OPEN;
  13   1      CLOSE:  PROCEDURE (AFTN,STATUS) EXTERNAL;
  14   2                  DECLARE (AFTN,STATUS) ADDRESS;
  15   2              END CLOSE;
  16   1      RQWAIT: PROCEDURE (exchange$address,time$limit) ADDRESS EXTERNAL;
  17   2                  DECLARE (exchange$address,time$limit) ADDRESS;
  18   2              END RQWAIT;
  19   1      READ:   PROCEDURE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
  20   2                  DECLARE (AFTN,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
  21   2              END READ;


  22   1      DECLARE OUT$EN          ADDRESS EXTERNAL;       /* Output Enable flag */
  23   1      DECLARE LOGGIN          ADDRESS EXTERNAL;
  24   1      DECLARE LOG$LEN         ADDRESS EXTERNAL;
  25   1      DECLARE status          ADDRESS;
  26   1      DECLARE dummy           ADDRESS;
```

```
                $EJECT
 27   1     LPRINT: PROCEDURE PUBLIC;
 28   2             DECLARE PRINTER          ADDRESS;
 29   2             DECLARE buffer(10000)    BYTE AT(.LOGGIN);
 30   2             DECLARE j                BYTE;
 31   2             DECLARE i                ADDRESS;

 32   2     PRINT:  PROCEDURE (buffer,length);
 33   3                 DECLARE (buffer,length) ADDRESS;

 34   3                 DO; DECLARE status      ADDRESS;
 36   4                     CALL WRITE (PRINTER,buffer,length,.status);

 37   4                     IF status<>0 THEN DO; CALL ERROR (status);
 40   5                                           CALL EXIT;
 41   5                                       END;
 42   4                 END /* block */;
 43   3             END PRINT;
```

```
                $EJECT
 44   2         CALL READ(1,.dummy,128,.dummy,.status); /* dummy read empty the buffer  */

 45   2         dummy = RQWAIT(.OUT$EN,0);              /* Wait for Output enable       */
 46   2         CALL OPEN(.PRINTER,.(':LP: '),2,0,.status);

 47   2                 i=0;                            /* Get first character address  */
 48   2         CONT:   IF (j:=buffer(i))>31
                         THEN CALL PRINT (.j,1);       /* PRINT the character          */
 50   2                 ELSE DO; DO CASE j ;
 52   4                     CALL PRINT (.('(NUL)'),5);  /* NULL */
 53   4                     CALL PRINT (.('(SOH)'),5);  /* SOH  */
 54   4                     CALL PRINT (.('(STX)'),5);  /* STX  */
 55   4                     CALL PRINT (.('(ETX)'),5);  /* ETX  */
 56   4                     CALL PRINT (.('(EOT)'),5);  /* EOT  */
 57   4                     CALL PRINT (.('(ENQ)'),5);  /* ENQ  */
 58   4                     CALL PRINT (.('(ACK)'),5);  /* ACK  */
 59   4                     CALL PRINT (.('(BEL)'),5);  /* BEL  */
 60   4                     CALL PRINT (.('(BS)'),4);   /* BS   */
 61   4                     CALL PRINT (.('(HT)'),4);   /* HT   */
 62   4                     CALL PRINT (.('(LF)'),4);   /* LF   */
 63   4                     CALL PRINT (.('(VT)'),4);   /* VT   */
 64   4                     CALL PRINT (.('(FF)'),4);   /* FF   */
 65   4                     IF buffer(i+1)=0AH
                              THEN DO;
 67   5                          CALL PRINT (.buffer(i),2);
 68   5                          i=i+1;
 69   5                          END;
                              ELSE
 70   4                          CALL PRINT (.('(CR)'),4); /* CR   */
 71   4                     CALL PRINT (.('(SO)'),4);   /* SO   */
 72   4                     CALL PRINT (.('(SI)'),4);   /* SI   */
 73   4                     CALL PRINT (.('(DLE)'),5);  /* DLE  */
 74   4                     CALL PRINT (.('(DC1)'),5);  /* DC1  */
 75   4                     CALL PRINT (.('(DC2)'),5);  /* DC2  */
 76   4                     CALL PRINT (.('(DC3)'),5);  /* DC3  */
 77   4                     CALL PRINT (.('(DC4)'),5);  /* DC4  */
 78   4                     CALL PRINT (.('(NAK)'),5);  /* NACK */
 79   4                     CALL PRINT (.('(SYN)'),5);  /* SYNC */
 80   4                     CALL PRINT (.('(ETB)'),5);  /* ETB  */
 81   4                     CALL PRINT (.('(CAN)'),5);  /* CAN  */
 82   4                     CALL PRINT (.('(EM)'),4);   /* EM   */
 83   4                     CALL PRINT (.('(SUB)'),5);  /* SUB  */
 84   4                     CALL PRINT (.('(ESC)'),5);  /* ESC  */
 85   4                     CALL PRINT (.('(FS)'),4);   /* FS   */
 86   4                     CALL PRINT (.('(GS)'),4);   /* GS   */
 87   4                     CALL PRINT (.('(RS)'),4);   /* RS   */
 88   4                     CALL PRINT (.('(US)'),4);   /* US   */
 89   4                 END /* DO CASE */;
 90   3                END /* ELSE    */;

 91   2                 i=i+1;                          /* Fetch next character address */
 92   2                 IF i<=LOG$LEN THEN GOTO CONT;   /* if there are more.           */

 94   2                 CALL PRINT (.(0DH,0AH),2);      /* Last line feed */

 95   2                 CALL CLOSE (PRINTER,.status);
```

```
 96   2              CALL EXIT;
 97   2          END LPRINT;
 98   1      END RESULT;
```

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SYNRES
OBJECT MODULE PLACED IN SYNRES.OBJ
COMPILER INVOKED BY:  PLM80 SYNRES.PLM PRINT(:LP:) PAGEWIDTH(132) XREF DEBUG TITLE('SYNRES') DATE(04.01.82) WORKFILES(:F0:,:F0:)


   1          SYN$RES:    DO;

                          $INCLUDE(:F0:TMSTR.ELT)
                =
   2    1      =  DECLARE     def$mres$syn    LITERALLY
                =  'STRUCTURE( link            ADDRESS,
                =              length          ADDRESS,
                =              type            BYTE,
                =              k$free$rec      ADDRESS,
                =              k$treat$rec     ADDRESS,
                =              k$prior$send    ADDRESS,
                =              k$norm$send     ADDRESS)';
                =
   3    1      =  DECLARE     def$bal         LITERALLY
                =  '           ptart           ADDRESS,
                =              kplein          ADDRESS,
                =              kvide           ADDRESS,
                =              lmin            ADDRESS';
                =
   4    1      =  DECLARE     def$hd$art$bal  LITERALLY
                =  '           link            ADDRESS,
                =              length          ADDRESS,
                =              type            BYTE,
                =              long            BYTE';
                =
   5    1      =  DECLARE     def$msg$ress    LITERALLY
                =  'STRUCTURE( msg$hdr,
                =              tres            BYTE)';
                =
   6    1      =  DECLARE     def$msg$bal     LITERALLY
                =  'STRUCTURE( def$hd$art$bal,
                =              tam(0)          BYTE)';
                =
   7    1      =  DECLARE     def$hdr$bloc    LITERALLY
                =  '           link            ADDRESS,
                =              length          ADDRESS,
                =              type            BYTE,
                =              home$exchange   ADDRESS,
                =              long            BYTE';
                =
   8    1      =  DECLARE     def$msg$bloc$rec LITERALLY
                =  'STRUCTURE( def$hdr$bloc,
                =              block (252)     BYTE)';
```

```
                =   $EJECT
      9     1   =   DECLARE     lart$cons       LITERALLY '72',
                =               lbal$cons       LITERALLY '1024',
                =               lart$impri      LITERALLY '133',
                =               lbal$impri      LITERALLY '512',
                =               lart$perfo      LITERALLY '80',
                =               lbal$perfo      LITERALLY '512';
                =
     10     1   =   DECLARE     t$cour$bal      LITERALLY '0',
                =               t$last$bal      LITERALLY '0',
                =               f$typ           LITERALLY '0',
                =               typ$b$libre     LITERALLY '0';  /*à définir*/
                =
     11     1   =   DECLARE     KC2$CONS        LITERALLY '62H',
                =               KC3             LITERALLY '20H',
                =               KETX            LITERALLY '03H',
                =               KETB            LITERALLY '17H';
```

```
                =   $EJECT
                =
                =   /* Messages types */
                =
     12    1    =   DECLARE typ$int              LITERALLY ' 1',     /* interrupt    */
                =           typ$trig$syn         LITERALLY '63',     /* Triggering   */
                =           typ$free$syn         LITERALLY '64',     /* free         */
                =           typ$alarm$syn        LITERALLY '65',     /* alarm        */
                =           typ$wrong$syn        LITERALLY '66',     /* wrong        */
                =           typ$err$blocked      LITERALLY '67',     /* error+blocked*/
                =           typ$stop$syn         LITERALLY '68',     /* stop request */
                =           typ$res$syn          LITERALLY '69',     /* restart req. */
                =           typ$rec$keyboard     LITERALLY '70',     /* keyboard req.*/
                =           typ$rec$print$ETB    LITERALLY '71',     /* printer req. */
                =           typ$rec$card$ETB     LITERALLY '72',     /* card reader  */
                =           typ$over$timed       LITERALLY '03',     /* time out RMX */
                =           typ$res$res          LITERALLY '73';     /* res. reserva.*/
                =
```

```
                $EJECT

                /* PUBLICS & EXTERNALS */

 13   1         DECLARE (tmmin,tmmout)          ADDRESS EXTERNAL /* Transmission automatons */; /* TMMRB.MAC */
 14   1         DECLARE (basin,basout,buflen)   ADDRESS PUBLIC   /* I/O buffers addresses   */; /* TMMRB.MAC */

 15   1         DECLARE final                   BYTE EXTERNAL    /* Automaton FINAL state   */; /*      "      */

 16   1         DECLARE (NOBnm1,NOBn)           BYTE PUBLIC      /* Record #                */; /*      "      */

 17   1         DECLARE (mode,svc,                               /* TMMRB.MAC : mode & svc  */  /*      "      */
                       perif1,perif2)          BYTE PUBLIC;     /*             peripherals */

 18   1         DECLARE (printer$OK,                             /* Peripheral control      */  /* ANY USER  */
                        puncher$OK)            BYTE PUBLIC;

 19   1         DECLARE (inlex0,
                       K$err$syn,
                       K$res$syn)              ADDRESS EXTERNAL;
```

```
                $EJECT

                /* RMX */

                $INCLUDE (:F0:MSG.ELT)
     20   1  =  DECLARE MSG$HDR LITERALLY '
             =      LINK     ADDRESS,
             =      LENGTH   ADDRESS,
             =      TYPE     BYTE,
             =      HOME$EXCHANGE   ADDRESS,
             =      RESP$EXCHANGE   ADDRESS';
             =

     21   1      RQSEND:
                     PROCEDURE (exchange$address,message$address) EXTERNAL;
     22   2         DECLARE (exchange$address,message$address) ADDRESS;
     23   2         END RQSEND ;

     24   1      RQACPT:
                     PROCEDURE (exchange$address) ADDRESS EXTERNAL;
     25   2         DECLARE  exchange$address ADDRESS;
     26   2         END RQACPT ;

     27   1      RQWAIT:
                     PROCEDURE (exchange$address,time$limit) ADDRESS EXTERNAL;
     28   2         DECLARE (exchange$address,time$limit) ADDRESS;
     29   2         END RQWAIT ;

     30   1      RQELVL:
                     PROCEDURE (interrupt$level) EXTERNAL;
     31   2         DECLARE interrupt$level BYTE;
     32   2         END RQELVL ;

     33   1      RQDLVL:
                     PROCEDURE (interrupt$level) EXTERNAL;
     34   2         DECLARE interrupt$level BYTE;
     35   2         END RQDLVL ;
```

```
                $EJECT

                /*  I/O Ports   */

 36   1         DECLARE status$8251        LITERALLY '0F5H',   /* usart 0   status port        */
                      data$8251          LITERALLY '0F4H',   /*           data   port        */
                      status$8259        LITERALLY '0FCH',   /* PIC       status port        */
                      mode$8253          LITERALLY '0F3H',   /* PIT #1    mode   port        */
                      counter$8253       LITERALLY '0F0H',   /*           counter port       */
                      system$PIC         LITERALLY '0FAH';   /* SYSTEM INT CONTROLLER        */
```

```
                $EJECT

                /*  Control words */

 37   1         DECLARE reset                LITERALLY '40H',    /* 8251       (pg 3-26)
                                                                 /*  40= 0 : no hunting mode
                                                                 /*      1 : INTERNAL reset
                                                                 /*      0 : RTS not forced
                                                                 /*      0 : FLAGS not reset
                                                                 /*      0 : NORMAL operation
                                                                 /*      0 : RECEIVE disable
                                                                 /*      0 : DTR not forced
                                                                 /*      0 : TRANSMIT disable    */
                         locked               LITERALLY '00H',    /* 8251 locked : all disable    */
                         hunting$mode         LITERALLY '96H',    /* 8251 hunting mode (pg 3-26
                                                                 /*  96= 1 : HUNTING mode set
                                                                 /*      0 : no INTERNAL RESET
                                                                 /*      0 : RTS not forced
                                                                 /*      1 : reset FLAGS
                                                                 /*      0 : NORMAL operation
                                                                 /*      1 : RECEIVE enable
                                                                 /*      1 : DTR forced
                                                                 /*      0 : TRANSMIT disable */
                         mode$instr$word      LITERALLY '98H',    /* 8251 Mode Instr. Word (pg 3-25
                                                                 /*  98= 1 : single SYNC character
                                                                 /*      0 : output SYNDET
                                                                 /*      0 : ODD parity
                                                                 /*      1 : PARITY enable
                                                                 /*      10: 7 bits char. length
                                                                 /*      00: (always)            */
                         sync$char            LITERALLY '16H',    /* SYNC char ASCII              */
                         cmd$instr$word       LITERALLY '92H',    /* 8251 Command Instr. Word (pg 3-26
                                                                 /*  92= 1 : HUNTING mode
                                                                 /*      0 : no INTERNAL reset
                                                                 /*      0 : RTS not forced
                                                                 /*      1 : all error FLAGS reset
                                                                 /*      0 : TxD not forced
                                                                 /*      0 : RECEIVE disable
                                                                 /*      1 : DTR forced
                                                                 /*      0 : TRANSMIT disable    */
                         transmit$enable      LITERALLY '33H';    /* 8251 (pg 3-26)
                                                                 /*  33= 0 : no HUNTING mode
                                                                 /*      0 : no INTERNAL reset
                                                                 /*      1 : RTS forced
                                                                 /*      1 : all FLAGS reset
                                                                 /*      0 : NORMAL operation
                                                                 /*      0 : RECEIVE disable
                                                                 /*      1 : DTR forced
                                                                 /*      1 : TRANSMIT enable     */
```

```
                $EJECT

 38   1     DECLARE mode$control$word   LITERALLY '36H',    /* 8253 (pg 3-7)
                                                           /*      00: COUNTER 0 selected
                                                           /*      11: READ LSB first, thenMSB
                                                           /*     x11: MODE 3
                                                           /*       0: binary counter      */
              counter$MSB                LITERALLY '01H',  /* PIT BAUD rate=4800 ==) N=256 */
              counter$LSB                LITERALLY '00H';  /* (MSB=1H, LSB=0H)             */

 39   1     DECLARE level$in            LITERALLY '00',   /* task interrupt level : in */
              level$out                  LITERALLY '01';   /*                      : out*/
                                                           /* TMMRB.MAC mode cont. words*/
 40   1     DECLARE all$forbidden       LITERALLY '0H',   /* 0 = ALL forbidden */
              SOH$autorised              LITERALLY '2',    /* 2 = SOH autorised */
              ACK$autorised              LITERALLY '1';    /*   = ACK autorised */

 41   1     DECLARE EOT                 LITERALLY '04H',  /* ASCII control characters */
              ACK                        LITERALLY '06H',
              NAK                        LITERALLY '15H',
              SEL                        LITERALLY '61H',
              DLE                        LITERALLY '10H',
              BLK$ETB                    LITERALLY '0',
              BLK$ETX                    LITERALLY '1',
              BLK$ERR                    LITERALLY '2',
              NOB$NAK                    LITERALLY '1',
              DLE$ACK                    LITERALLY '2',    /* Automaton final state     */
              DLE$NAK                    LITERALLY '3',    /* control char. (TMMRB.MAC) */
              BLOC$ACCEPTE               LITERALLY 'NOT final';
```

```
                $EJECT

                /* Messages declarations */

 42   1         DECLARE (msg$base,                              /* Messages addresses */
                        int$msg$base)               ADDRESS;

 43   1         DECLARE msg     BASED msg$base      def$msg$bloc$rec;   /* Message structure */
 44   1         DECLARE int$msg BASED int$msg$base  def$msg$bloc$rec;   /* Int. msg structure */
 45   1         DECLARE mress                       def$mres$syn;       /* Resource reservation */

 46   1         DECLARE (svc$msg1,                              /* Service messages buffers */
                        svc$msg2)           BYTE;               /* Two bytes only           */

 47   1         DECLARE (fictif$msg$base,                       /* Other buffers addresses  */
                        kb$wait$msg$base,
                        CR$wait$msg$base)   ADDRESS;

 48   1         DECLARE per$fictif$rec      BYTE;
 49   1         DECLARE (bloque,famille)    BYTE;
```

```
                $EJECT

 50   1     DECLARE TRUE                LITERALLY '0FFH',
                    FALSE               LITERALLY '00H',
                    FOREVER             LITERALLY 'WHILE TRUE';
 51   1     DECLARE BOOLEAN             LITERALLY 'BYTE';
 52   1     DECLARE dummy               BYTE;

 53   1     DECLARE fictif              LITERALLY '60H',    /* Some peripherals */
                    keyboard            LITERALLY '62H',    /* & their equi-    */
                    puncher             LITERALLY '63H',    /* valents in se-   */
                    printer             LITERALLY '61H';    /*  ction  mode     */

 54   1     DECLARE CR$polled           BOOLEAN;
```

```
                $EJECT

                /* PROCEDURES */

 55   1         ENABLE$OUTPUT: PROCEDURE;
 56   2             final=-1;                              /* Reset final state    */
 57   2             CALL RQELVL (level$out);               /* Autorise IT (out)    */
 58   2             OUTPUT(status$8251)=transmit$enable;   /* 8251 transmiting     */
 59   2             END ENABLE$OUTPUT;

 60   1         SEND$SVC$MSG: PROCEDURE (svc1,svc2,svc$mode);
 61   2             DECLARE   (svc1,svc2,svc$mode) BYTE;
 62   2             svc=TRUE;                              /* Service message flag */
 63   2             svc$msg1=svc1;                         /* Message buffers      */
 64   2             svc$msg2=svc2;                         /* (1 or 2 bytes)       */
 65   2             basout=.svc$msg1;                      /* Serv. msg address    */
 66   2             mode=svc$mode;                         /* Set required mode    */
 67   2             CALL ENABLE$OUTPUT;                    /* 8251 status          */
 68   2             END SEND$SVC$MSG;

 69   1         HUNTING: PROCEDURE;
 70   2             final=-1;
 71   2             mode=all$forbidden;
 72   2             OUTPUT(status$8251)=hunting$mode;
 73   2             dummy=INPUT(data$8251);
 74   2             END HUNTING;

 75   1         UPDATE$NOB: PROCEDURE;
 76   2             NOBnm1=NOBn;                           /* Memorize last NOBn*/
 77   2             NOBn=(NOBn+1) AND 47H;                 /* modulo 8H +40H    */
 78   2             END UPDATE$NOB;
```

```
                $EJECT

 79   1     RENVOI$BLOC:
                PROCEDURE(ad$msg);
 80   2          DECLARE ad$msg ADDRESS;
 81   2          DECLARE msg BASED ad$msg DEF$MSG$BLOC$REC;
 82   2          CALL RQSEND(msg.home$exchange,.msg);
 83   2          END RENVOI$BLOC;

 84   1     BLOCAGE:
                PROCEDURE;
 85   2          bloque=TRUE;
 86   2          IF famille=1
                  THEN
 87   2            mress.type=typ$alarm$syn;          /* Send error msg of*/
                  ELSE
 88   2            mress.type=typ$wrong$syn;          /* one of those types */
 89   2          CALL RQSEND (.K$err$syn,.mress);
 90   2          CALL HUNTING;
 91   2          END BLOCAGE;

 92   1     RENVOI$INT$MSG:
                PROCEDURE(type);
 93   2          DECLARE type BYTE;
 94   2          DECLARE msg BASED int$msg$base STRUCTURE(MSG$HDR);
 95   2          msg.type=type;
 96   2          CALL RQSEND(msg.resp$exchange,.msg);
 97   2          END RENVOI$INT$MSG;

 98   1     LIBERATE$RES$BLOCKS:
                PROCEDURE;
 99   2          IF per$fictif$rec
                  THEN
100   2            CALL RENVOI$BLOC(fictif$msg$base);
101   2          IF kb$wait$msg$base()0
                  THEN
102   2            CALL RENVOI$BLOC(kb$wait$msg$base);
103   2          IF CR$wait$msg$base()0
                  THEN
104   2            CALL RENVOI$BLOC(CR$wait$msg$base);
105   2          bloque=2;
106   2          END LIBERATE$RES$BLOCKS;
```

```
                $EJECT

107   1     SELECTING:
                PROCEDURE BYTE;
108   2         IF per$fictif$rec
                  THEN DO;
110   3           per$fictif$rec=FALSE;
111   3           CALL RQSEND (mress.K$free$rec,fictif$msg$base);
112   3           END;
113   2         perif2=keyboard;
                /* Which selection is beeing asked for ? */
114   2         DO CASE final-60H;
115   3           perif1,perif2 = fictif;
116   3           DO;
117   4             IF NOT printer$OK
                    THEN
118   4               RETURN FALSE;
119   4             perif1=printer;
120   4             END;
121   3           perif1=keyboard;
122   3           DO;
123   4             IF NOT puncher$OK
                    THEN
124   4               RETURN FALSE;
125   4             perif1 = puncher;
126   4             END;
127   3           END;    /*do case*/
128   2         RETURN(msg$base:=RQACPT(mress.K$free$rec))()0;
129   2         END SELECTING;

130   1     SEND$SEL$NAK:
                PROCEDURE;      /* Block not free */
131   2         CALL SEND$SVC$MSG (SEL,NAK,all$forbidden);
132   2         END SEND$SEL$NAK;
```

```
                $EJECT

133   1     POURSUITE$SELECTING:
                PROCEDURE BYTE;
134   2         IF perif1=fictif
                  THEN DO;
136   3           fictif$msg$base=msg$base;
137   3           RETURN(per$fictif$rec:=TRUE);
138   3           END;
139   2         msg.type=ROL(perif1-61h,1)+typ$rec$print$etb+final;
140   2         msg.long=buflen;
141   2         CALL RQSEND(mress.k$treat$rec,msg$base);        /* Send block for treatment */
142   2         IF final=BLK$ETB                                /* Test if ETX or ETB :     */
                  THEN DO;                                    /* If ETB test keyboard     */
144   3           IF kb$wait$msg$base<>0
                    THEN
145   3             RETURN FALSE;
146   3           IF (msg$base:=RQACPT(mress.K$prior$send))<>0
                    THEN DO;
148   4             kb$wait$msg$base=msg$base;
149   4             RETURN FALSE;
150   4             END;
151   3           RETURN(msg$base:=RQACPT(mress.K$free$rec))<>0;
152   3           END;
153   2         IF perif1=printer
                  THEN
154   2           printer$ok=FALSE;
155   2         IF perif1=puncher
                  THEN
156   2           puncher$ok=FALSE;
157   2         RETURN FALSE;
158   2         END POURSUITE$SELECTING;
```

```
                $EJECT

 159   1      TEST$USER:
                PROCEDURE BYTE;
 160   2          IF kb$wait$msg$base()0
                   THEN
 161   2              msg$base=kb$wait$msg$base;
                   ELSE
 162   2          IF (CR$polled) AND (CR$wait$msg$base()0)
                   THEN
 163   2              msg$base=CR$wait$msg$base;
                   ELSE
 164   2          IF (msg$base:=RQACPT(mress.K$prior$send))()0
                   THEN DO;
 166   3              kb$wait$msg$base = msg$base;
 167   3              END;
                   ELSE
 168   2          IF CR$polled
                   THEN
 169   2              IF (msg$base:=RQACPT(mress.K$norm$send))()0
                       THEN
 170   2                  CR$wait$msg$base=msg$base;
                       ELSE
 171   2                      RETURN FALSE;
                   ELSE
 172   2              RETURN FALSE;
 173   2          RETURN TRUE;
 174   2          END TEST$USER;

 175   1      POLLING:
                PROCEDURE BYTE;
 176   2          IF final=40H        /* 40 H */
                   THEN DO;
 178   3              msg$base=fictif$msg$base;
 179   3              RETURN per$fictif$rec;
 180   3              END;
 181   2          IF final=41H        /* 41 H */
                   THEN
 182   2              CR$polled=TRUE;
 183   2          RETURN TEST$USER;
 184   2          END POLLING;
```

```
                $EJECT

185   1     POURSUITE$POLLING:
                PROCEDURE BYTE;
186   2         IF BLOC$ACCEPTE
                  THEN DO;
188   3           CALL UPDATE$NOB;
189   3           CALL RENVOI$BLOC(msg$base);
190   3           IF per$fictif$rec AND msg$base=fictif$msg$base
                    THEN
191   3             RETURN(per$fictif$rec:=FALSE);
192   3           IF kb$wait$msg$base()0
                    THEN
193   3             kb$wait$msg$base=0;
                    ELSE
194   3             CR$wait$msg$base=0;
195   3           IF final=DLE$ACK
                    THEN
196   3             RETURN FALSE;
                    ELSE
197   3             RETURN TEST$USER;
198   3           END;
199   2         RETURN final=NOB$NAK;
200   2         END POURSUITE$POLLING;

201   1     SEND$BLOCK:
                PROCEDURE;
202   2         mode=ACK$autorised;
203   2         svc=FALSE;
204   2         basout=.msg.block(0);                 /* Skip RMX control bytes */
205   2         buflen=msg.long;                      /* length of blocks       */
206   2         CALL ENABLE$OUTPUT;                   /* Send the message       */
207   2         END SEND$BLOCK;

208   1     POLLING$END:
                PROCEDURE;
209   2         CR$polled = FALSE;
210   2         CALL SEND$SVC$MSG (EOT,dummy,all$forbidden);
211   2         END POLLING$END;
```

```
                $EJECT
212   1         SYN$RES:
                    PROCEDURE PUBLIC;
213   2             DO FOREVER;
214   3                 OUTPUT (status$8251)=locked;              /* Initialization    */
215   3                 per$fictif$rec=FALSE;
216   3                 svc=TRUE;                                 /* Service msg autor */
217   3                 printer$OK,puncher$OK,CR$polled=FALSE;
218   3                 CALL RQDLVL(level$in);
219   3                 CALL RQDLVL(level$out);
220   3                 mress.type=typ$res$res;                   /* MRESS typ resource reservation */
221   3                 CALL RQSEND (.K$res$syn,.mress);          /* Gerant becomes available      */
222   3                 DO WHILE RQWAIT (.inlex0,0)().mress;      /* Wait for triggering request */
223   4                     END;
224   3                 IF mress.type=typ$trig$syn
                        THEN DO;
226   4                     fictif$msg$base=0;
227   4                     kb$wait$msg$base=0;
228   4                     CR$wait$msg$base=0;
229   4                     DISABLE;                                  /* Disable IT */
230   4                     OUTPUT (mode$8253)=mode$control$word;     /* Init of 8253 */
231   4                     OUTPUT (counter$8253)=counter$LSB;        /* Baud rate */
232   4                     OUTPUT (counter$8253)=counter$MSB;
233   4                     OUTPUT (status$8251)=00H;                 /* initialisation of 8251 */
234   4                     OUTPUT (status$8251)=00H;
235   4                     OUTPUT (status$8251)=00H;
236   4                     OUTPUT (status$8251)=reset;               /* Init of 8251 (pg 3-27) */
237   4                     OUTPUT (status$8251)=mode$instr$word;
238   4                     OUTPUT (status$8251)=sync$char;
239   4                     OUTPUT (status$8251)=cmd$instr$word;
240   4                     ENABLE;
241   4                     bloque=FALSE;
242   4                     CALL HUNTING;                             /* Continue in sleeping state */
243   4                     CALL RQELVL(level$in);
```

```
                $EJECT

 244   4                DO WHILE bloque()2;
 245   5                   int$msg$base=RQWAIT(.inlex0,0);          /* Wait until IT        */
 246   5                   IF int$msg.type=typ$int                  /* RQDLVL if IT from    */
                           THEN DO;
 248   6                       famille=ROL(final,3) AND 3;
 249   6                       CALL RQDLVL(level$out);              /* automaton (TMMRB.MAC)*/
 250   6                       IF bloque
                               THEN DO;
 252   7                           IF famille=3                     /*(SEL)(NAK) if SELECTION */
                                   THEN
 253   7                               CALL SEND$SEL$NAK;
                                   ELSE
 254   7                               IF famille=2                 /*(EOT) if POLLING */
                                       THEN
 255   7                                   CALL POLLING$END;
                                       ELSE
 256   7                                   CALL HUNTING;            /* in case neither SEL nor POL */
 257   7                           END;
                               ELSE
 258   6                           IF mode=all$forbidden
                               THEN DO;
 260   7                           NOBn=(NOBnm1:=40H)+1;
 261   7                           DO CASE famille;
 262   8                               CALL HUNTING;
 263   8                               CALL BLOCAGE;
 264   8                               IF POLLING
                                       THEN
 265   8                                   CALL SEND$BLOCK;
                                       ELSE
 266   8                                   CALL POLLING$END;
 267   8                               IF SELECTING
                                       THEN DO;
 269   9                                   basin=.msg.block(0);                          /* Skip RMX control bytes */
 270   9                                   CALL SEND$SVC$MSG (SEL,ACK,SOH$autorised);    /* Send (SEL)(ACK) */
 271   9                                   END;
                                       ELSE
 272   8                                   CALL SEND$SEL$NAK;
 273   8                               END;
 274   7                           END;
                               ELSE
 275   6                           IF mode=SOH$autorised
                               THEN DO;
 277   7                           IF final=BLK$ERR
                                   THEN
 278   7                               CALL SEND$SVC$MSG (NOBnm1,NAK,SOH$autorised);
                                   ELSE
 279   7                               IF final=EOT
                                       THEN DO;
 281   8                                   IF((perif1()fictif) or NOT per$fictif$rec)
                                           THEN
 282   8                                       CALL RQSEND (mress.K$free$rec,msg$base);
 283   8                                   CALL HUNTING;
 284   8                                   END;
                                       ELSE
 285   7                                   IF final(=BLK$ETX
```

```
                                    THEN DO;
 287   8                                IF POURSUITE$SELECTING
                                        THEN DO;
 289   9                                    basin=.msg.block(0);
 290   9                                    CALL SEND$SVC$MSG (NOBn,ACK,SOH$autorised);
 291   9                                    CALL UPDATE$NOB;
 292   9                                    END;
                                        ELSE
 293   8                                    CALL SEND$SVC$MSG (DLE,ACK,all$forbidden);
 294   8                                END;
 295   7                            ELSE DO;
 296   8                                CALL ROSEND(mress.K$free$rec,msg$base);
 297   8                                CALL BLOCAGE;
 298   8                                END;
 299   7                            END;
 300   6                        ELSE DO;     /*ACK autorised*/
 301   7                            IF final(=DLE$NAK
                                    THEN DO;
 303   8                                IF POURSUITE$POLLING
                                        THEN
 304   8                                    CALL SEND$BLOCK;
                                        ELSE
 305   8                                    CALL POLLING$END;
 306   8                                END;
 307   7                            ELSE DO;
 308   8                                CR$polled=FALSE;
 309   8                                CALL BLOCAGE;
 310   8                                END;
 311   7                            END;
 312   6                        END;     /*fin taitement message It*/
                              ELSE
 313   5                        IF bloque
                               THEN DO;     /*message interne*/
 315   6                            IF int$msg$base().mress
                                    THEN
 316   6                                CALL RENVOI$INT$MSG(typ$err$blocked);
                                    ELSE
 317   6                                IF int$msg.type=typ$res$syn
                                        THEN
 318   6                                    bloque=FALSE;
                                        ELSE
 319   6                                    CALL LIBERATE$RES$BLOCKS;
 320   6                                END;
                               ELSE
 321   5                            IF int$msg.type=typ$free$syn
                                    THEN DO;
 323   6                                CALL LIBERATE$RES$BLOCKS;
 324   6                                CALL RENVOI$INT$MSG(typ$stop$syn);
 325   6                                END;
                                    /*autres messages perdus*/
                               END;     /*do while*/
 327   4                    END;
 328   3                END;     /*do forever*/
 329   2            END SYN$RES;
 330   1        END SYN$RES;
```

```
ASM80 INTSYN.ASM PRINT(:LP:) TITLE('04.01.82') PAGEWIDTH(132) MOD85 DEBUG XREF MACROFILE


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0         INTSYN   PAGE    1
04.01.82

  LOC  OBJ         LINE        SOURCE STATEMENT

                     1
                     2 ;*********************************************************************;
                     3 ;                                                                     ;
                     4 ;       TMM/RB : Synchronous messages transmission line               ;
                     5 ;                                                                     ;
                     6 ;       Two procédures are defined under IT :                         ;
                     7 ;               - TMMIN  : Receiver automaton                         ;
                     8 ;               - TMMOUT : Sender   automaton                         ;
                     9 ;                                                                     ;
                    10 ;*********************************************************************;
                    11
                    12         NAME    INTSYN
                    13
                    14         CSEG
                    15
                    16         PUBLIC  TMMIN,TMMOUT            ; Receiver & Sender procedures under IT
                    17         PUBLIC  FINAL                   ; Receiver FINAL state
                    18                                         ; 0FFH          not finished yet
                    19                                         ; 0 mode=1      NOBACK mode=2   ETB
                    20                                         ; 1             NOBNAK          ETX
                    21                                         ; 2             DLEACK          errored bloc
                    22                                         ; 3             DLENAK
                    23                                         ; 4             EOT
                    24                                         ; 02FH          SELBEL
                    25                                         ; 040H à 042H   POLLING
                    26                                         ; 060H à 063H   SECTION
                    27
                    28         EXTRN   INLEXO                  ; associated interrupt exchange
                    29         EXTRN   RQISND,RQENDI           ; RMX procedures
                    30
                    31         EXTRN   BASIN,BASOUT            ; I/O data blocks addresses
                    32         EXTRN   BUFLEN                  ; & its length
                    33
                    34         EXTRN   NOBn                    ; number of the [to be] sent block : NOB (n)
                    35         EXTRN   NOBnm1                  ; number of the last correct block received : NOB (n-1)
                    36
                    37         EXTRN   SVC                     ; Service message ind.  1 : service msg
                    38                                         ;                       0 : non service
                    39         EXTRN   MODE                    ; Input mode indicator  1 : ack autorized
                    40                                         ;                       2 : block   "
                    41
                    42         EXTRN   PERIF1,PERIF2           ; autorized peripherals
                    43         PUBLIC  LOGGIN                  ; BUFFER
                    44         PUBLIC  LOGLEN                  ; BUFFER LENGTH
                    45 $EJECT
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

                     46 ;ASCII conventions :
                     47
 0001                48 SOH      EQU     01H
 0002                49 STX      EQU     02H
 0003                50 ETX      EQU     03H
 0004                51 EOT      EQU     04H
 0005                52 ENQ      EQU     05H
 0006                53 ACK      EQU     06H
 0007                54 BEL      EQU     07H
 0010                55 DLE      EQU     10H
 0015                56 NAK      EQU     15H
 0016                57 SYN      EQU     16H
 0017                58 ETB      EQU     17H
 001E                59 RS       EQU     1EH
                     60
 0020                61 C3       EQU     20H
 0041                62 POL      EQU     41H
 0061                63 SEL      EQU     61H
 007F                64 PAD      EQU     7FH
                     65
                     66 ;        Port addresses :
                     67
 00F4                68 DATA     EQU     0F4H                ; data port
 00F5                69 STATUS   EQU     0F5H                ; status port
                     70
                     71 ;        Particular variables used :
                     72
 0018                73 ERROR    EQU     00011000B           ; Cfr USART doc.
 0008                74 PARITY   EQU     00001000B           ;       "
 00F0                75 FALSE    EQU     0F0H
                     76
 0000                77 BLKETB   EQU     0
 0001                78 BLKETX   EQU     1
 0003                79 INCFST   EQU     3                   ;incorrect final state
 0000                80 NOBACK   EQU     0
 0001                81 NOBNAK   EQU     1
 0002                82 ACKDLE   EQU     2                   ;DLE ACK
 0003                83 NAKDLE   EQU     3                   ;DLE NAK
 0001                84 ACKMD    EQU     1
 0002                85 ACKBLK   EQU     2
 0010                86 LOCK     EQU     00010000B           ; Cfr RHUNT without Receive enable & Hunting mode
 0096                87 RHUNT    EQU     10010110B           ; 1 : Hunting mode
                     88                                      ; 0 : No internal reset
                     89                                      ; 0 : No RTS
                     90                                      ; 1 : Error reset
                     91                                      ; 0 : No break
                     92                                      ; 1 : Receive enable
                     93                                      ; 1 : DTR forced
                     94                                      ; 0 : Transmit desable
                     95
                     96 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     97
                     98 ;       Variables location :
                     99
  0000 2F27      C   100 SENSTA: DW      SINIT                   ; Sender initial state
  0002 0329      C   101 RECSTA: DW      RINIT                   ; Receiver initial state
  0004 00            102 COUNT:  DB      0                       ; Byte counter
  0005               103 BCC:    DS      1                       ; BCC check sum
  0006               104 FINAL:  DS      1                       ; Receiver final state
  0007               105 LENGTH: DS      1                       ; One record length
  0008               106 RCASE:  DS      1
                     107
  0009 FF            108 LC2:    DB      255D                    ; if C2 = 60H
  000A 85            109         DB      133D                    ;       61H
  000B 48            110         DB       72D                    ;       62H
  000C 50            111         DB       80D                    ;       63H
                     112
                     113 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     114
                     115 ;Macro's definition :
                     116
                     117 PUSHRG   MACRO                          ; Push all registers on pile
                     118 ;        PUSH     PSW      ;;2          ; Status word : it's done within POLLIT (pg.1)
  -                  119          PUSH     B        ;;3
  -                  120          PUSH     D        ;;4
  -                  121          PUSH     H        ;;5
                     122          ENDM
                     123
                     124
                     125 POPRG    MACRO                          ; POP all registers from pile
  -                  126          POP      H        ;;4
  -                  127          POP      D        ;;3
  -                  128          POP      B        ;;2
  -                  129          POP      PSW      ;;1          ; return addr is left on pile
                     130          ENDM
                     131
                     132 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     133
  000D 0F00      C   134 LOGADR: DW      LOGGIN
  000F               135 LOGGIN: DS      10000D                  ; LOGGIN BUFFER
  271F 0000          136 LOGLEN: DW      0                       ; LOGGIN BUFFER LENGTH
                     137
                     138
                     139
                     140 LOGM    MACRO                           ; BUFFERIZES THE CHAR OF THE LINE
  -                  141         PUSH    H
  -                  142         LHLD    LOGADR
  -                  143         MOV     M,A
  -                  144         INX     H
  -                  145         SHLD    LOGADR
  -                  146         LHLD    LOGLEN
  -                  147         INX     H
  -                  148         SHLD    LOGLEN
  -                  149         POP     H
                     150         ENDM
                     151
                     152
                     153
                     154 OUTM    MACRO
  -                  155         LOGM
  -                  156         OUT     DATA
                     157         ENDM
                     158
                     159 LFCR    MACRO
  -                  160         PUSH    H
  -                  161         LHLD    LOGADR
  -                  162         MVI     M,0DH ; CR CHAR
  -                  163         INX     H
  -                  164         MVI     M,0AH ; LF CHAR
  -                  165         INX     H
  -                  166         SHLD    LOGADR
  -                  167         LHLD    LOGLEN
  -                  168         INX     H
  -                  169         INX     H
  -                  170         SHLD    LOGLEN
  -                  171         POP     H
                     172         ENDM
                     173
                     174
                     175
                     176
                     177 INM     MACRO
  -                  178         IN      DATA
  -                  179         LOGM
                     180         ENDM
                     181
                     182 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     183 ;*****************************************
                     184 ;                                       ;
                     185 ;       TMMOUT  :       SENDER          ;
                     186 ;                                       ;
                     187 ;*****************************************;
                     188
                     189
                     190 TMMOUT:
                     191         PUSHRG                  ;;5     ; save reg on pile
                     192+;       PUSH    PSW     ;;2             ; Status word : it's done within POLLIT (pg.1)
  2721 C5            193+        PUSH    B
  2722 D5            194+        PUSH    D
  2723 E5            195+        PUSH    H
                     196
  2724 110500    C   197         LXI     D,BCC                   ; DE points to BCC
  2727 019D28    C   198         LXI     B,SCONT                 ; get next sender return address
  272A C5            199         PUSH    B               ;;6     ; & push it on pile
                     200
  272B 2A0000    C   201         LHLD    SENSTA                  ; restitute sender last state
  272E E9            202         PCHL                            ; & jump to its entry point
                     203
                     204 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     205
                     206 ;Messages configuration :
                     207 ;
                     208 ; - Service : (SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SERVICE) (SYN) (SYN)
                     209 ;
                     210 ; - Data    : (SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) (SEL) (NOBn) (STX) (.../.../...) (BCC) (SYN) (SYN)
                     211 ;                                                  ^-------------------------^
                     212 ;                                                            BCC compute
                     213
                     214 SINIT:
  272F 3EFA          215         MVI     A,250                   ; Sender INITial state
  2731 320400    C   216         STA     COUNT                   ; init counter
                     217
                     218 ; Common beginning to all messages :
                     219
                     220 SSYN6:
  2734 3E16          221         MVI     A,SYN                   ; fetch (SYN) char.
                     222         OUTM                            ; send it 6 times
                     223+        LOGM
  2736 E5            224+        PUSH    H
  2737 2A0D00    C   225+        LHLD    LOGADR
  273A 77            226+        MOV     M,A
  273B 23            227+        INX     H
  273C 220D00    C   228+        SHLD    LOGADR
  273F 2A1F27    C   229+        LHLD    LOGLEN
  2742 23            230+        INX     H
  2743 221F27    C   231+        SHLD    LOGLEN
  2746 E1            232+        POP     H
  2747 D3F4          233+        OUT     DATA
  2749 210400    C   234         LXI     H,COUNT                 ; update counter
  274C 34            235         INR     M                       ; untill it's 0
  274D CA5427    C   236         JZ      SDMSG                   ; after (SYN) send the message
                     237
  2750 213427    C   238         LXI     H,SSYN6                 ; else send (SYN) once more
  2753 C9            239         RET                     ;;5     ; & return
                     240
                     241
                     242 SDMSG:
  2754 216027    C   243         LXI     H,SSERV                 ; since there are more service than data msg.
  2757 3A0000    E   244         LDA     SVC                     ; fetch message type
  275A B7            245         ORA     A                       ; test if 0
  275B C0            246         RNZ                     ;;5     ; SVC = 1 if service
  275C 219C27    C   247         LXI     H,SDATA                 ;       0 if non-service
  275F C9            248         RET                     ;;5     ; & return any case
                     249 $EJECT
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

                    250 ;*************** Send service msg ************************************
                    251
                    252 ; Service msg types :   1.      (EOT)
                    253 ;                       2.      (NOB(n)) (ACK)
                    254 ;                       3.      (NOB(n-1)) (NAK)
                    255 ;                       4.      (DLE) (ACK)
                    256 ;                       5.      (SEL) (ACK)
                    257 ;                       6.      (SEL) (NAK)
                    258
                    259
                    260
                    261 SSERV:
 2760 2A0000    E   262          LHLD    BASOUT                  ; buffer addr.
 2763 7E            263          MOV     A,M                     ; fetch service byte
                    264          OUTM                            ; send it
                    265+         LOGM
 2764 E5            266+         PUSH    H
 2765 2A0D00    C   267+         LHLD    LOGADR
 2768 77            268+         MOV     M,A
 2769 23            269+         INX     H
 276A 220D00    C   270+         SHLD    LOGADR
 276D 2A1F27    C   271+         LHLD    LOGLEN
 2770 23            272+         INX     H
 2771 221F27    C   273+         SHLD    LOGLEN
 2774 E1            274+         POP     H
 2775 D3F4          275+         OUT     DATA
 2777 FE04          276          CPI     EOT                     ; check if (EOT) ?
 2779 CA9827    C   277          JZ      ESERVS                  ; yes : go with "end service sending"
                    278
 277C 218027    C   279          LXI     H,OTHER                 ; no  : prepare for next char.
 277F C9            280          RET                     ;;5
                    281
                    282
                    283 OTHER:
 2780 2A0000    E   284          LHLD    BASOUT                  ; point to char buffer addr.
 2783 23            285          INX     H                       ; point to next char.
 2784 7E            286          MOV     A,M                     ; fetch it
                    287          OUTM
                    288+         LOGM
 2785 E5            289+         PUSH    H
 2786 2A0D00    C   290+         LHLD    LOGADR
 2789 77            291+         MOV     M,A
 278A 23            292+         INX     H
 278B 220D00    C   293+         SHLD    LOGADR
 278E 2A1F27    C   294+         LHLD    LOGLEN
 2791 23            295+         INX     H
 2792 221F27    C   296+         SHLD    LOGLEN
 2795 E1            297+         POP     H
 2796 D3F4          298+         OUT     DATA
                    299
                    300
                    301 ESERVS:
 2798 214B28    C   302          LXI     H,SSYN2                 ; end with sending with 2 (SYN)
 279B C9            303          RET                     ;;5
                    304 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     305 ;*************** Send data messages *********************************
                     306
                     307 ; Data msg types :      (block header) ::= (SOH) (SEL) (NOB(n)) (STX)
                     308 ;                       (data   block) ::= (CHAR) [(CHAR)..]
                     309 ;                       (block   BCC ) ::= (BCC)
                     310
                     311
                     312
                     313 SDATA:
                     314 SSOH:
  279C 3E01          315         MVI     A,SOH                   ; send (SOH)
                     316         OUTM
                     317+        LOGM
  279E E5            318+        PUSH    H
  279F 2A0D00    C   319+        LHLD    LOGADR
  27A2 77            320+        MOV     M,A
  27A3 23            321+        INX     H
  27A4 220D00    C   322+        SHLD    LOGADR
  27A7 2A1F27    C   323+        LHLD    LOGLEN
  27AA 23            324+        INX     H
  27AB 221F27    C   325+        SHLD    LOGLEN
  27AE E1            326+        POP     H
  27AF D3F4          327+        OUT     DATA
  27B1 21B527    C   328         LXI     H,SSEL                  ; send after (SEL)
  27B4 C9            329         RET                     ;;5
                     330
                     331
                     332 SSEL:
  27B5 3E61          333         MVI     A,SEL                   ; send (SEL)
                     334         OUTM
                     335+        LOGM
  27B7 E5            336+        PUSH    H
  27B8 2A0D00    C   337+        LHLD    LOGADR
  27BB 77            338+        MOV     M,A
  27BC 23            339+        INX     H
  27BD 220D00    C   340+        SHLD    LOGADR
  27C0 2A1F27    C   341+        LHLD    LOGLEN
  27C3 23            342+        INX     H
  27C4 221F27    C   343+        SHLD    LOGLEN
  27C7 E1            344+        POP     H
  27C8 D3F4          345+        OUT     DATA
  27CA 21CE27    C   346         LXI     H,SNOB                  ; send after (NOBn)
  27CD C9            347         RET                     ;;5
                     348
                     349 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     350
                     351 SNOB:
  27CE 3A0000   E    352         LDA     NOBn                ; send NOB(n)
                     353         OUTM
                     354+        LOGM
  27D1 E5            355+        PUSH    H
  27D2 2A0D00   C    356+        LHLD    LOGADR
  27D5 77            357+        MOV     M,A
  27D6 23            358+        INX     H
  27D7 220D00   C    359+        SHLD    LOGADR
  27DA 2A1F27   C    360+        LHLD    LOGLEN
  27DD 23            361+        INX     H
  27DE 221F27   C    362+        SHLD    LOGLEN
  27E1 E1            363+        POP     H
  27E2 D3F4          364+        OUT     DATA
  27E4 EE61          365         XRI     SEL                 ; BCC computed with the first char. after (SOH)
  27E6 EB            366         XCHG                        ; HL points now to BCC (ignore DE)
  27E7 77            367         MOV     M,A                 ; save real BCC
  27E8 21EC27   C    368         LXI     H,SSTX              ; send after (STX)
  27EB C9            369         RET                 ;;5
                     370
                     371
                     372 SSTX:
  27EC 3E02          373         MVI     A,STX               ; send (STX)
                     374         OUTM
                     375+        LOGM
  27EE E5            376+        PUSH    H
  27EF 2A0D00   C    377+        LHLD    LOGADR
  27F2 77            378+        MOV     M,A
  27F3 23            379+        INX     H
  27F4 220D00   C    380+        SHLD    LOGADR
  27F7 2A1F27   C    381+        LHLD    LOGLEN
  27FA 23            382+        INX     H
  27FB 221F27   C    383+        SHLD    LOGLEN
  27FE E1            384+        POP     H
  27FF D3F4          385+        OUT     DATA
  2801 EB            386         XCHG                        ; HL points to BCC addr.
  2802 AE            387         XRA     M                   ; computing BCC
  2803 77            388         MOV     M,A                 ; save it
  2804 210828   C    389         LXI     H,SCHAR             ; send after a (char)
  2807 C9            390         RET                 ;;5
                     391
                     392 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                    393
                    394 SCHAR:
  2808 2A0000   E   395         LHLD    BASOUT              ; buffer address
  280B 7E           396         MOV     A,M                 ; get a char.
                    397         OUTM                        ; send it
                    398+        LOGM
  280C E5           399+        PUSH    H
  280D 2A0D00   C   400+        LHLD    LOGADR
  2810 77           401+        MOV     M,A
  2811 23           402+        INX     H
  2812 220D00   C   403+        SHLD    LOGADR
  2815 2A1F27   C   404+        LHLD    LOGLEN
  2818 23           405+        INX     H
  2819 221F27   C   406+        SHLD    LOGLEN
  281C E1           407+        POP     H
  281D D3F4         408+        OUT     DATA
  281F 23           409         INX     H                   ; point to next char
  2820 220000   E   410         SHLD    BASOUT              ; save this addr.
  2823 EB           411         XCHG
  2824 AE           412         XRA     M                   ; compute BCC
  2825 77           413         MOV     M,A                 ; save it
  2826 210000   E   414         LXI     H,BUFLEN            ; dec buffer length
  2829 35           415         DCR     M
  282A 210828   C   416         LXI     H,SCHAR             ; presuming a msg is composed by several bytes
  282D C0           417         RNZ                 ;;5     ; ret now if more char to send
  282E 213228   C   418         LXI     H,SBCC              ; else send BCC after
  2831 C9           419         RET                 ;;5
                    420
                    421
                    422 SBCC:
  2832 EB           423         XCHG                        ; HL points to BCC
  2833 7E           424         MOV     A,M                 ; send BCC
                    425         OUTM
                    426+        LOGM
  2834 E5           427+        PUSH    H
  2835 2A0D00   C   428+        LHLD    LOGADR
  2838 77           429+        MOV     M,A
  2839 23           430+        INX     H
  283A 220D00   C   431+        SHLD    LOGADR
  283D 2A1F27   C   432+        LHLD    LOGLEN
  2840 23           433+        INX     H
  2841 221F27   C   434+        SHLD    LOGLEN
  2844 E1           435+        POP     H
  2845 D3F4         436+        OUT     DATA
  2847 214B28   C   437         LXI     H,SSYN2             ; end with two (SYN)
  284A C9           438         RET                 ;;5
                    439 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     440 ;*************** Common end to all messages ***************************
                     441
                     442
                     443
                     444 SSYN2:
  284B 3E16          445         MVI     A,SYN                   ; send first (SYN)
                     446         OUTM
                     447+        LOGM
  284D E5            448+        PUSH    H
  284E 2A0D00    C   449+        LHLD    LOGADR
  2851 77            450+        MOV     M,A
  2852 23            451+        INX     H
  2853 220D00    C   452+        SHLD    LOGADR
  2856 2A1F27    C   453+        LHLD    LOGLEN
  2859 23            454+        INX     H
  285A 221F27    C   455+        SHLD    LOGLEN
  285D E1            456+        POP     H
  285E D3F4          457+        OUT     DATA
  2860 216428    C   458         LXI     H,SSYN1                 ; left one (SYN) to send
  2863 C9            459         RET                     ;;5
                     460
                     461 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     462
                     463 SSYN1:
  2864 3E16          464         MVI     A,SYN                   ; send last (SYN)
                     465         OUTM
                     466+        LOGM
  2866 E5            467+        PUSH    H
  2867 2A0D00    C   468+        LHLD    LOGADR
  286A 77            469+        MOV     M,A
  286B 23            470+        INX     H
  286C 220D00    C   471+        SHLD    LOGADR
  286F 2A1F27    C   472+        LHLD    LOGLEN
  2872 23            473+        INX     H
  2873 221F27    C   474+        SHLD    LOGLEN
  2876 E1            475+        POP     H
  2877 D3F4          476+        OUT     DATA
                     477         LFCR
  2879 E5            478+        PUSH    H
  287A 2A0D00    C   479+        LHLD    LOGADR
  287D 360D          480+        MVI     M,0DH ; CR CHAR
  287F 23            481+        INX     H
  2880 360A          482+        MVI     M,0AH ; LF CHAR
  2882 23            483+        INX     H
  2883 220D00    C   484+        SHLD    LOGADR
  2886 2A1F27    C   485+        LHLD    LOGLEN
  2889 23            486+        INX     H
  288A 23            487+        INX     H
  288B 221F27    C   488+        SHLD    LOGLEN
  288E E1            489+        POP     H
  288F E1            490         POP     H               ;;5     ; cancel fictive return addr.
                     491
  2890 212F27    C   492         LXI     H,SINIT                 ; restore initial state
  2893 220000    C   493         SHLD    SENSTA
  2896 21682A    C   494         LXI     H,RCONT                 ; force receiver restore addr.
  2899 E5            495         PUSH    H               ;;6
  289A C3C328    C   496         JMP     HUNT                    ; force receiver HUNTING mode
                     497
                     498
                     499 ;*****************************************************************
                     500
                     501 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     502
                     503 SCONT:
  289D 220000    C   504          SHLD    SENSTA                  ; save next entry point in Sender automaton
                     505 RESTR:
  28A0 CD0000    E   506          CALL    RQENDI          ;;7     ; RMX call
                     507                                  ;;6
                     508          POPRG                   ;;1     ; restore all registers
  28A3 E1            509+         POP     H
  28A4 D1            510+         POP     D
  28A5 C1            511+         POP     B
  28A6 F1            512+         POP     PSW
  28A7 FB            513          EI                              ; re-enable IT
  28A8 C9            514          RET                             ; & get out
                     515 $EJECT
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                   516 ;****************************************
                   517 ;                                        ;
                   518 ;       TMMIN   :       RECEIVER         ;
                   519 ;                                        ;
                   520 ;****************************************
                   521
                   522
                   523 TMMIN:
                   524             PUSHRG                       ; save all reg.
                   525+;           PUSH    PSW     ;;2          ; Status word : it's done within POLLIT (pg.1)
28A9 C5            526+            PUSH    B
28AA D5            527+            PUSH    D
28AB E5            528+            PUSH    H
                   529
28AC 21682A    C   530             LXI     H,RCONT              ; force it as return addr.
28AF E5            531             PUSH    H           ;;6
                   532
28B0 DBF5          533             IN      STATUS               ; fetch USART 544 state
28B2 E618          534             ANI     ERROR                ; any error ?
28B4 CAD528    C   535             JZ      RNOERR               ; no  : skip to "NO-ERROR"
                   536
28B7 E608          537             ANI     PARITY               ; test for parity error ?
28B9 CAD028    C   538             JZ      RWRONG               ; no parity error : skip but remember wrong char!
                   539
28BC DBF4          540             IN      DATA                 ; else, get the char even if it's bad
28BE FE7F          541             CPI     PAD                  ; test if it's a (PAD) ?
28C0 C2D028    C   542             JNZ     RWRONG               ; no : it's another char (but a bad one)
                   543                                          ; else (if (PAD)) return hunting :
                   544 HUNT:
28C3 3E96          545             MVI     A,RHUNT              ; return hunting mode
28C5 D3F5          546             OUT     STATUS               ; & force USART to hunt
28C7 3E00          547             MVI     A,00H                ; reset the char counter
28C9 320400    C   548             STA     COUNT
28CC 210329    C   549             LXI     H,RINIT              ; & automaton to INITIAL state
28CF C9            550             RET                          ; return anyway.
                   551
                   552
                   553 RWRONG:
28D0 3EF0          554             MVI     A,FALSE              ; force a wrong character
28D2 C3F128    C   555             JMP     ANALYS               ; continue as if nothing happened, knowing per-
                   556                                          ; fectly it'll cause a wrong BCC
                   557
                   558 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     559
                     560 RNOERR:
                     561          INM                          ; get the char from USART
  28D5 DBF4          562+         IN       DATA
                     563+         LOGM
  28D7 E5            564+         PUSH     H
  28D8 2A0D00    C   565+         LHLD     LOGADR
  28DB 77            566+         MOV      M,A
  28DC 23            567+         INX      H
  28DD 220D00    C   568+         SHLD     LOGADR
  28E0 2A1F27    C   569+         LHLD     LOGLEN
  28E3 23            570+         INX      H
  28E4 221F27    C   571+         SHLD     LOGLEN
  28E7 E1            572+         POP      H
  28E8 FE16          573          CPI      SYN                  ; test for (SYN) ?
  28EA C2F128    C   574          JNZ      ANALYS               ; no  : go further in analysing char
  28ED C1            575          POP      B                    ; yes : skip all encountered (SYN)
  28EE C3A028    C   576          JMP      RESTR                ; and get out of the automaton
                     577 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     578 ANALYS:
  28F1 5F            579         MOV     E,A                 ; save char
  28F2 210500    C   580         LXI     H,BCC               ; point to BCC (first time it's of no use-
  28F5 AE            581         XRA     M                   ; compute it    (because not initialized)
  28F6 77            582         MOV     M,A                 ; & save it
                     583
  28F7 7B            584         MOV     A,E                 ; re-get the char
  28F8 210400    C   585         LXI     H,COUNT             ; update char. counter
  28FB 34            586         INR     M                   ; test if ) 256 ?
  28FC CAC328    C   587         JZ      HUNT                ; yes : return hunting !
                     588
  28FF 2A0200    C   589         LHLD    RECSTA              ; else: restitute RECeiver STAte
  2902 E9            590         PCHL                        ; & jump to its entry point
                     591
                     592
                     593 ;*****************************************************************
                     594
                     595
                     596 RINIT:
  2903 FE04          597         CPI     EOT                 ; test if (EOT)
  2905 CAB829    C   598         JZ      REOT                ; jump if (EOT) received
                     599
  2908 FE61          600         CPI     SEL
  290A CAE529    C   601         JZ      RSEL                ; jump if (SEL) received
                     602
  290D FE41          603         CPI     POL
  290F CAE929    C   604         JZ      RPOL                ; jump if (POL) received
                     605                                     ; else it's neither (EOT) nor (SOL) or (POL)
  2912 3A0000    E   606         LDA     MODE                ; so test for autorised mode : block or ACK ?
  2915 FE01          607         CPI     ACKMD               ; test first for ACK mode ?
  2917 CAED29    C   608         JZ      RBACK               ; yes : receive ACK
                     609
  291A FE02          610         CPI     ACKBLK              ; test either for block ?
  291C C2C328    C   611         JNZ     HUNT                ; nothing is convenient : return hunting
                     612
                     613 $EJECT
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

                    614 ;**************** Treatening with a block ***************************
                    615
                    616
                    617 RBLK:                                   ; receiving block autorised
                    618 RSOH:
 291F 3E01          619         MVI     A,SOH                   ; supposed to receive first (SOH)
 2921 BB            620         CMP     E                       ; check it ?
 2922 C2C328    C   621         JNZ     HUNT                    ; return hunting if not !
                    622
 2925 212929    C   623         LXI     H,RBPOL                 ; next time supposed to receive (SEL)
 2928 C9            624         RET
                    625
                    626
                    627 RBPOL:
 2929 FE61          628         CPI     SEL                     ; (SEL) received ?
 292B C25A2A    C   629         JNZ     BLKERR                  ; no  : jump to "block error"
 292E 320500    C   630         STA     BCC                     ; initialize BCC with first char. after (SOH)
 2931 213529    C   631         LXI     H,RNOBn                 ; else, next time supposed to receive NOB(n)
 2934 C9            632         RET
                    633
                    634
                    635 RNOBn:
 2935 210000    E   636         LXI     H,NOBn                  ; presumed received block #
 2938 BE            637         CMP     M                       ; yes ?
 2939 C25A2A    C   638         JNZ     BLKERR                  ; no ! incorrect block #
 293C 214029    C   639         LXI     H,RSTX                  ; else, next time supposed to receive (STX)
 293F C9            640         RET
                    641
                    642
                    643 RSTX:
 2940 FE02          644         CPI     STX                     ; (STX) received ?
 2942 C25A2A    C   645         JNZ     BLKERR                  ; no : incorrect block structure
 2945 214929    C   646         LXI     H,TPERIF                ; else, next time, analyse periph. type (C2)
 2948 C9            647         RET
                    648
                    649 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     650
                     651 RC2:                                    ; wait for receiving C2
                     652 TPERIF:
  2949 210000    E   653         LXI     H,PERIF1                ; check for autorised peripheral
  294C BE            654         CMP     M
  294D CA5529    C   655         JZ      C2                      ; OK, go on
  2950 23            656         INX     H                       ; else : INC HL
  2951 BE            657         CMP     M                       ; check the second one
  2952 C25A2A    C   658         JNZ     BLKERR                  ; it's an error !
                     659
                     660
                     661 C2:
  2955 21A9FF    C   662         LXI     H,LC2-60H               ; fetch L(C2) conversion table
  2958 1600          663         MVI     D,0                     ; null D reg so DE = C2
  295A 19            664         DAD     D                       ; HL (-- HL+DE (points now to L(C2)
  295B 7E            665         MOV     A,M                     ; fetch L(C2)
  295C 320700    C   666         STA     LENGTH                  ; & save it as record length
                     667
  295F 2A0000    E   668         LHLD    BASIN                   ; point to last next buffer char
  2962 73            669         MOV     M,E                     ; put the char. in buffer
  2963 23            670         INX     H                       ; & update pointer before
  2964 220000    E   671         SHLD    BASIN                   ; saving it
  2967 216B29    C   672         LXI     H,RC3                   ; next time receive C3
  296A C9            673         RET                     ;;5
                     674
                     675 $EJECT
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

                    676
                    677 RC3:
 296B FE20          678         CPI     C3                      ; received C3 ?
 296D C25A2A    C   679         JNZ     BLKERR                  ; no  : block error
 2970 C37F29    C   680         JMP     BUFFER                  ; else: buffer it
                    681
                    682
                    683 RDATA:
 2973 FEF0          684         CPI     FALSE                   ; check if rec. char is not the dump one (i.e. FALSE)
 2975 CA5A2A    C   685         JZ      BLKERR                  ; else jump
                    686
 2978 210700    C   687         LXI     H,LENGTH                ; update block length
 297B 35            688         DCR     M                       ; test if not beyond record length
 297C CA5A2A    C   689         JZ      BLKERR                  ; yes: error because (RS) not seen
                    690
                    691 BUFFER:
 297F 2A0000    E   692         LHLD    BASIN                   ; point to buffer
 2982 77            693         MOV     M,A                     ; buffer the char.
 2983 23            694         INX     H                       ; update pointer
 2984 220000    E   695         SHLD    BASIN                   ; & save it
 2987 0601          696         MVI     B,BLKETX
 2989 FE03          697         CPI     ETX                     ; check if (ETX)
 298B CA9F29    C   698         JZ      RETX                    ; yes : jump
 298E 0600          699         MVI     B,BLKETB
 2990 FE17          700         CPI     ETB                     ; check if (ETB)
 2992 CA9F29    C   701         JZ      RETB
                    702
 2995 FE1E          703         CPI     RS                      ; check if (RS)
 2997 217329    C   704         LXI     H,RDATA                 ; presuming a message is composed of several characters
 299A C0            705         RNZ                     ;;5     ; return if neither (ETX),(ETB) nor (RS)
 299B 214929    C   706         LXI     H,RC2                   ; else wait for rec. another record within THIS block
 299E C9            707         RET                     ;;5
                    708
                    709 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     710
                     711 RETX:
                     712 RETB:
  299F 78            713         MOV     A,B
  29A0 320600    C   714         STA     FINAL                   ; restitute allready final state
  29A3 3A0400    C   715         LDA     COUNT
  29A6 D604          716         SUI     4                       ; header
  29A8 320000    E   717         STA     BUFLEN
                     718
  29AB 21AF29    C   719         LXI     H,RBCC                  ; wait for receiving BCC for any correction
  29AE C9            720         RET                         ;;5
                     721
                     722 RBCC:
  29AF 3A0500    C   723         LDA     BCC                     ; correct BCC ?
  29B2 B7            724         ORA     A
  29B3 CABB29    C   725         JZ      RECEND                  ; yes : END with RECeiver work
                     726
                     727 ERREND:
  29B6 3E03          728         MVI     A,INCFST                ; else: force INCorrect Final STate
                     729
                     730 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     731
                     732 EOWRK:                                  ; End Of WoRK entry point
                     733 REOT:
  29B8 320600    C   734          STA     FINAL                  ; final state ((ETX),(EOB),(ENQ),(EOT))
                     735 RECEND:
  29BB 3E10          736          MVI     A,LOCK                 ; block the 8251
  29BD D3F5          737          OUT     STATUS
  29BF 210329    C   738          LXI     H,RINIT                ; reset receiver in INITIAL state
  29C2 220200    C   739          SHLD    RECSTA                 ; save RECeiver STAte
  29C5 010000    E   740          LXI     B,INLEXO               ; IT channel exchange address
                     741          LFCR
  29C8 E5            742+         PUSH    H
  29C9 2A0D00    C   743+         LHLD    LOGADR
  29CC 360D          744+         MVI     M,0DH ; CR CHAR
  29CE 23            745+         INX     H
  29CF 360A          746+         MVI     M,0AH ; LF CHAR
  29D1 23            747+         INX     H
  29D2 220D00    C   748+         SHLD    LOGADR
  29D5 2A1F27    C   749+         LHLD    LOGLEN
  29D8 23            750+         INX     H
  29D9 23            751+         INX     H
  29DA 221F27    C   752+         SHLD    LOGLEN
  29DD E1            753+         POP     H
                     754
  29DE CD0000    E   755          CALL    RQISND          ;;7    ; send IT to superior level
                     756                                  ;;6
  29E1 E1            757          POP     H               ;;5    ; retrieve RET address (not used any more)
  29E2 C3A328    C   758          JMP     RESTR+3                ; & restore registers
                     759
                     760
                     761 RSEL:
  29E5 21392A    C   762          LXI     H,RC2SEL               ; wait for receiving peripheral
  29E8 C9            763          RET                     ;;5    ; selection code C2(SEL)
                     764
                     765
                     766 RPOL:
  29E9 214D2A    C   767          LXI     H,RC2POL               ; wait for receiving peripheral
  29EC C9            768          RET                     ;;5    ; pooling code C2(POL)
                     769 $EJECT
```

```
 LOC  OBJ          LINE        SOURCE STATEMENT

                    770 ;*************** Threatening with "ACK" messages **********************
                    771
                    772
                    773
                    774 RBACK:
 29ED 3A0000    E   775         LDA     NOBn                    ; receiving the "sent block #" ACK
 29F0 BB            776         CMP     E                       ; is it the good one ?
 29F1 CA052A    C   777         JZ      OKNOBn                  ; yes : jump
                    778
 29F4 3A0000    E   779         LDA     NOBnm1                  ; no : resending last block ?
 29F7 BB            780         CMP     E
 29F8 CA0C2A    C   781         JZ      OKnm1                   ; yes: jump
                    782
 29FB 3E10          783         MVI     A,DLE                   ; else : sending (DLE) ?
 29FD BB            784         CMP     E
 29FE C2C328    C   785         JNZ     HUNT                    ; no : there's an error somewhere, return hunting !
                    786
 2A01 21252A    C   787         LXI     H,DLEACK                ; else wait for (DLE) ACK or NAK
 2A04 C9            788         RET                     ;;5
                    789
                    790 OKNOBn:
 2A05 3E01          791         MVI     A,NOBNAK                ; 80H (= final state (= 87H
 2A07 0606          792         MVI     B,ACK                   ; wait for receiving (ACK)
 2A09 C3102A    C   793         JMP     OKSERV                  ; point to RACK:
                    794
                    795 OKnm1:
 2A0C 3E01          796         MVI     A,NOBNAK                ; C0H (= final state (= C7H
 2A0E 0615          797         MVI     B,NAK                   ; wait for receiving (NAK)
                    798 OKSERV:
 2A10 320600    C   799         STA     FINAL
 2A13 78            800         MOV     A,B
 2A14 320800    C   801         STA     RCASE
 2A17 211B2A    C   802         LXI     H,RACK
 2A1A C9            803         RET                     ;;5
                    804
                    805 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     806
                     807 RACK:
  2A1B 3A0800    C   808         LDA     RCASE               ; will be either in (ACK),(NAK) or (ENQ)
  2A1E BB            809         CMP     E
  2A1F C2C328    C   810         JNZ     HUNT                ; if wrong char received, return hunting
  2A22 C3BB29    C   811         JMP     RECEND              ; else RECeiver ENDs its work
                     812                                     ; final state having allready beeing saved
                     813
                     814 DLEACK:
  2A25 FE06          815         CPI     ACK                 ; is it an (ACK) ?
  2A27 C22F2A    C   816         JNZ     NOTACK              ; no : jump to if it's a (NAK)
  2A2A 3E02          817         MVI     A,ACKDLE
  2A2C C3B829    C   818         JMP     EOWRK
                     819
                     820 NOTACK:
  2A2F FE15          821         CPI     NAK                 ; is it then a (NAK) ?
  2A31 C2C328    C   822         JNZ     HUNT                ; no : so return hunting mode
  2A34 3E03          823         MVI     A,NAKDLE
  2A36 C3B829    C   824         JMP     EOWRK
                     825
                     826
                     827 $EJECT
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                   828 RC2SEL:
2A39 FE2F          829         CPI     2FH                 ; check if correct C2(SEL)
2A3B CA482A    C   830         JZ      SELOK               ;   must be within (2F,60H,61H,62H,63H)
2A3E FE60          831         CPI     60H                 ; ( 60H ?
2A40 DAC328    C   832         JC      HUNT                ; incorrect C2(SEL) force return in hunting mode
2A43 FE64          833         CPI     64H                 ; ) 64H ?
2A45 D2C328    C   834         JNC     HUNT
                   835
                   836 POLOK:                              ; (See below in RC2POL)
                   837 SELOK:
2A48 0605          838         MVI     B,ENQ               ; wait for receiving an (ENQ)
2A4A C3102A    C   839         JMP     OKSERV
                   840
                   841 RC2POL:
2A4D FE40          842         CPI     40H                 ; ( 40H ?
2A4F DAC328    C   843         JC      HUNT                ; C2(POL) must be within [40H..42H]
2A52 FE43          844         CPI     43H                 ; any error forcing return in hunting mode
2A54 D2C328    C   845         JNC     HUNT
2A57 C3482A    C   846         JMP     POLOK               ; saving final state & waiting for (ENQ)$EJECT
                   847
                   848 BLKERR:
2A5A FE03          849         CPI     ETX                 ; when block error occurs somewhere,
2A5C CAB629    C   850         JZ      ERREND              ; skip all char. untill an (ETX) or
2A5F FE17          851         CPI     ETB                 ; an (ETB) appears;
2A61 CAB629    C   852         JZ      ERREND              ; when reached, go End Of Work
2A64 E1            853         POP     H           ;;5     ; don't need return address till it's RCONT:
2A65 215A2A    C   854         LXI     H,BLKERR            ; next state
                   855
                   856
                   857 RCONT:
2A68 220200    C   858         SHLD    RECSTA              ; RECeiver STAte saving
2A6B C3A028    C   859         JMP     RESTR               ; go restore reg. & return
                   860
                   861 $EJECT
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

                     862
                     863          END
```

PUBLIC SYMBOLS

FINAL	C 0006	LOGGIN	C 000F	LOGLEN	C 271F	TMMIN	C 28A9	TMMOUT	C 2721						

EXTERNAL SYMBOLS

BASIN	E 0000	BASOUT	E 0000	BUFLEN	E 0000	INLEXO	E 0000	MODE	E 0000	NOBN	E 0000	NOBNM1	E 0000		
PERIF1	E 0000	PERIF2	E 0000	RQENDI	E 0000	RQISND	E 0000	SVC	E 0000						

USER SYMBOLS

ACK	A 0006	ACKBLK	A 0002	ACKDLE	A 0002	ACKMD	A 0001	ANALYS	C 28F1	BASIN	E 0000	BASOUT	E 0000		
BCC	C 0005	BEL	A 0007	BLKERR	C 2A5A	BLKETB	A 0000	BLKETX	A 0001	BUFFER	C 297F	BUFLEN	E 0000		
C2	C 2955	C3	A 0020	COUNT	C 0004	DATA	A 00F4	DLE	A 0010	DLEACK	C 2A25	ENQ	A 0005		
EOT	A 0004	EOWRK	C 29B8	ERREND	C 29B6	ERROR	A 0018	ESERVS	C 2798	ETB	A 0017	ETX	A 0003		
FALSE	A 00F0	FINAL	C 0006	HUNT	C 28C3	INCFST	A 0003	INLEXO	E 0000	INM	+ 0000	LC2	C 0009		
LENGTH	C 0007	LFCR	+ 0004	LOCK	A 0010	LOGADR	C 000D	LOGGIN	C 000F	LOGLEN	C 271F	LOGM	+ 0002		
MODE	E 0000	NAK	A 0015	NAKDLE	A 0003	NOBACK	A 0000	NOBN	E 0000	NOBNAK	A 0001	NOBNM1	E 0000		
NOTACK	C 2A2F	OKNM1	C 2A0C	OKNOBN	C 2A05	OKSERV	C 2A10	OTHER	C 2780	OUTM	+ 0003	PAD	A 007F		
PARITY	A 0008	PERIF1	E 0000	PERIF2	E 0000	POL	A 0041	POLOK	C 2A48	POPRG	+ 0001	PUSHRG	+ 0000		
RACK	C 2A1B	RBACK	C 29ED	RBCC	C 29AF	RBLK	C 291F	RBPOL	C 2929	RC2	C 2949	RC2POL	C 2A4D		
RC2SEL	C 2A39	RC3	C 296B	RCASE	C 0008	RCONT	C 2A68	RDATA	C 2973	RECEND	C 29BB	RECSTA	C 0002		
REOT	C 29B8	RESTR	C 28A0	RETB	C 299F	RETX	C 299F	RHUNT	A 0096	RINIT	C 2903	RNOBN	C 2935		
RNOERR	C 28D5	RPOL	C 29E9	RQENDI	E 0000	RQISND	E 0000	RS	A 001E	RSEL	C 29E5	RSOH	C 291F		
RSTX	C 2940	RWRONG	C 28D0	SBCC	C 2832	SCHAR	C 2808	SCONT	C 289D	SDATA	C 279C	SDMSG	C 2754		
SEL	A 0061	SELOK	C 2A48	SENSTA	C 0000	SINIT	C 272F	SNOB	C 27CE	SOH	A 0001	SSEL	C 27B5		
SSERV	C 2760	SSOH	C 279C	SSTX	C 27EC	SSYN1	C 2864	SSYN2	C 284B	SSYN6	C 2734	STATUS	A 00F5		
STX	A 0002	SVC	E 0000	SYN	A 0016	TMMIN	C 28A9	TMMOUT	C 2721	TPERIF	C 2949				

ASSEMBLY COMPLETE, NO ERRORS

```
ASM80 POLL.SIM PRINT(:LP:) TITLE('04.01.82') PAGEWIDTH(132) MOD85 DEBUG XREF MACROFILE


ISIS-II 8080/8085 MACRO ASSEMBLER, V4.0         POLLIT   PAGE    1
04.01.82

  LOC  OBJ          LINE        SOURCE STATEMENT

                      1 ;*************************************************************************
                      2 ;
                      3 ; ROUTINE DE NIVEAU 7 DU SIC   (System Interrupt Controler)
                      4 ; EFFECTUE LE POLLING DE L'ISR DU LIC (Local  Interrupt Controler)
                      5 ; AFIN DE DETERMINER QUEL NIVEAU D'IT E/S DE RxrdyO,TxrdyO
                      6 ; A ETE SOLLICITE, PUIS SE BRANCHE A LA ROUTINE INIT CONCERNEE
                      7 ;
                      8 ;*************************************************************************
                      9
                     10
                     11          NAME    POLLIT
                     12
                     13
  00FB               14 LICPO    EQU     0FBH    ; LIC PORT CMMD 0
  00FA               15 LICP1    EQU     0FAH    ;      .        1
                     16
                     17
  000A               18 OCW      EQU     0AH     ; MOT DE POLLING (LECTURE ISR)
                     19
                     20          ASEG
  0038               21          ORG     38H
  0038 F5            22          PUSH    PSW     ; Le registre A est utilise dans POLLIT
  0039 C30000    C   23          JMP     POLL
                     24
                     25          EXTRN   TMMIN,TMMOUT
                     26
                     27          CSEG
                     28
                     29 POLL:
  0000 3E0A          30          MVI     A,OCW   ; LECTURE DE L'ISR (In Service Register)
  0002 D3FB          31          OUT     LICPO   ; DU LIC
  0004 DBFB          32          IN      LICPO
                     33
  0006 1F            34          RAR
  0007 DA0000    E   35          JC      TMMIN
                     36
  000A 1F            37          RAR
  000B DA0000    E   38          JC      TMMOUT
                     39
  000E F3            40          DI                      ; desable interrupts
  000F 76            41          HLT                     ; and crashes the system ...
                     42
                     43          END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS
TMMIN  E 0000    TMMOUT E 0000

USER SYMBOLS
LICPO  A 00FB    LICP1  A 00FA    OCW    A 000A    POLL   C 0000    TMMIN  E 0000    TMMOUT E 0000
```

```
  LOC  OBJ          LINE        SOURCE STATEMENT

ASSEMBLY COMPLETE,  NO ERRORS
```

Premier exemple de session :

(Les lignes précédées d'un point "." sont les messages de commandes que la station envoie au maître.)

```
.G
 ??FTOO 2312810935     VO2O4 PO2
 ??     CIT1           YOUR NAME IS INTY   DAY:OOO4,HOUR:OO18,MIN:OO14
.D SC,LP
 II SOPC  SC INFO 18x15x39
          PBID ST CA  FN  FH
          INFO LP O1  O1  OO
 .D SLP1
 ??FTOO 2312810935     VO2O4 PO2
```

```
(SYN) (EOT)          ← LE MAITRE ATTEND UNE REPONSE DE LA STATION
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN) ——→  LA STATION N'A RIEN A LUI TRANSMETTRE
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b G (ETB) O (SYN) (SYN) ——→  LA STATION TRANSMET LE MESSAGE "G"
(SYN) (SYN) (SYN) (SYN) (SYN) à (NAK) ——→  LE MAITRE A MAL REÇU LE MESSAGE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b G (ETB) O (SYN) (SYN) ——→  LA STATION RE-ESSAIE
(SYN) (DLE) (ACK) ——→  LE MAITRE ACCEPTE MAIS FAIT UNE SUSPENSION
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)  ← LE MAITRE AVERTI LA STATION QU'IL VA LUI ENVOYER UN MSG
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN) ——→ LA STATION ACCEPTE
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b ??FT00  2312810935      V0204 P02 (ETB) o ——→  LE MSG "?? FT00 2312810935 V0204 P02" est BIEN REÇU
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)                                    LE MAITRE ATTEND UN
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)   MSG DE LA STATION
(SYN) (EOT)
A (SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)  ← LE MAITRE AVERTI QU'IL VA ENVOYER UN MSG
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b ??    CIT1      YOUR NAME IS INTY      DAY:0004,HOUR:0018,MIN:0014 (ETB):
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
```

⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SOH⟩ aA ⟨STX⟩ b D SC, LP ⟨ETB⟩ 3 ⟨SYN⟩ ⟨SYN⟩ ⟶ LA STATION ENVOIE UN MSG
⟨SYN⟩ ⟨DLE⟩ ⟨ACK⟩ ⟶ LE MESSAGE EST ACCEPTÉ PAR LE MAITRE.
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩
⟨SYN⟩ ⟨EOT⟩
⟨SYN⟩ AB ⟨ENQ⟩
⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨SYN⟩ ⟨EOT⟩ ⟨SYN⟩ ⟨SYN⟩

LE MAITRE ATTEND UN MESSAGE EN PROVENANCE DE LA STATION

(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (EOT)
ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b]] SOPC SC INFO 18*15*39 (ETB) a ⟶ MESSAGE EN PROVENANCE DU MAITRE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b PBID ST CA FN FH (ETB) k ⟶ MESSAGE EN PROVENANCE DU MAITRE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)

```
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b          INFO LP 01  01  00 (ETB) e  ⟶  MESSAGE EN PROVENANCE DU MAITRE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) , (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) , (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b D SLP1 (ETB) m (SYN) (SYN)  ⟶  1ère ESSAI D'ENVOI DE LA STATION
(SYN) (SYN) (SYN) (SYN) (SYN) à (NAK)  ⟵  REFUS DU MAITRE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b D SLP1 (ETB) m (SYN) (SYN)  ⟶  2è ESSAI D'ENVOI
(SYN) (SYN) (DLE) (ACK)  ⟵  ACCEPTE PAR LE MAITRE
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b ??FT00  2312810935      V0204 P02 (ETB) c
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
```

Deuxième exmple de session :

```
.D SC,LP
 II SOPC  SC INFO  18x34x49
          PBID ST CA  FN  FH
          INFO LP 01  01  00
.D SCR1
 II SOPC  PB   18x35x05
    INFO OF TELE UP  SCR1 PDF INCR    CIT1 ST 00
.D
 II SOPC ER     6
.?D
 II SOPC ER     6
```

(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b D SC, LP (ETB) 3 (SYN) (SYN) ——→ DEC EN VOSE PAR
(SYN) (DLE) (ACK) CA STATION
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)

(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(US) (US) A (SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b]] SOPC SC INFO 18*34*49 (ETB) e ⟶
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b PBID ST CA FN FH (ETB) k
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)

```
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b        INFO LP 01  01  00 (ETB) e
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b D SCR1 (ETB) ' (SYN) (SYN) —>
(SYN) (SYN) (DLE) (ACK)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b JJ SOPC  PB   18*35*05 (ETB) '
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (EOT)
(SYN) (EOT)
```

NOTIFICATIONS ENVOYÉES PAR LE DISTRIBUTEUR

NOTIFICATION ENVOYÉE PAR LA STATION

(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b INFO OF TELE UP SCR1 PDF INCR CIT1 ST 00 (ETB) (DC2)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)

(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(US) (US) / (SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b D (ETB) 3 (SYN) (SYN)
(SYN) (SYN) (DLE) (ACK)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b]] SOPC ER 6 (ETB) Y
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)

```
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b ?D (ETB) (FF) (SYN) (SYN)
(SYN) (SYN) (DLE) (ACK)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN) (SYN)
(SYN) (EOT)
(SYN) ab (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) a (ACK) (SYN) (SYN)
(SYN) (SYN) (SYN) (SYN) (SYN) (SOH) aA (STX) b JJ SOPC ER     6 (ETB) Y
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (DLE) (ACK) (SYN) (SYN)
(SYN) (SYN) (EOT)
(SYN) (EOT)
(SYN) AB (ENQ)
(SYN) (SYN) (SYN) (SYN) (SYN) (SYN) (EOT) (SYN)
```

ANNEXE C

Mise en oeuvre d'un réseau local

LE MODELE PROCEDURAL

OBJECTIF DU MODELE PROCEDURAL

L'objectif du modèle procédural est de fournir des définitions précises des interfaces intercouches de l'architecture retenue du réseau. Il est nécessaire de spécifier les notations qui vont être utilisées dans la suite.

Les notations reposent sur l'utilisation du langage Pascal. Chaque interface est décrit comme un ensemble de procédures et/ou de variables, communes aux couches qu'il sert, qui fournissent conjointement les seules interactions valides entre couches de l'architecture.
Notons que la description des interfaces en Pascal est une technique de spécification, et n'implique nullement qu'ils doivent être réalisés sous forme de logiciel. De même, il faut souligner que c'est le comportement de chaque couche et non sa structure interne qui doit correspondre aux specifications. Dans ce but, les détails internes au modèle procédural ne sont utiles que pour aider à spécifier de manière claire et précise ce comportement.

Néanmoins, quelques observations doivent être faites quant à l'utilisation du langage Pascal pour la description du modèle :

a) Certaines limitations du langage ont été contournées en vue de simplifier les spécifications :

1. Les éléments du programme (Variables,procédures, etc.) sont présentés par groupe logique, selon une séquence "Bottom-Up". Certaines restrictions de présentation ont donc été délaissées au profit de la lisibilité.

2. Les constructions de processus et de cycle (PROCESS & CYCLE) du Pascal Concurrent ont été introduites pour indiquer les sites d'activités autonomes concurrentes. Utilisé ici, un processus est simplement une procédure, sans paramètres, dont l'exécution commence au "début du temps" plutôt qu'à la suite d'un appel de procédure. Un cycle représente le corps du processus qui est exécuté indéfiniment.

3. En ce qui concerne les trames (Frame), on a contourné l'absence de tableau (ARRAY) à dimensions variables en traitant chaque trame comme si elle était de longueur fixe et unique, ce qui n'a jamais été spécifié. En fait, bien sûr, la taille d'une trame dépend de la taille de sa zone de données (DataField) ; il faut donc

considérer la valeur de la "pseudo-constante" FrameSize (Taille de la trame) comme évoluant à long terme.

b) Le modèle n'utilise aucune primitive pour une synchronisation explicite entre processus. Au contraire, toutes les interactions entre processus sont réalisées par de subtiles manipulations de variables partagées par ces processus.

APERCU DES INTERFACES INTERCOUCHES

L'architecture choisie se décompose en quatre couches. Chaque couche peut demander un service à la couche qui lui est directement inférieure via un interface, appelé interface inter-couche.

On décrit dans la suite chaque interface :

1. Interface Physique - Liaison de données

L'interface est constitué de 2 procédures et d'une variable :

PROCEDURE TransmitByte(ByteParam:Byte)
Envoyer le byte contenu dans ByteParam sur la ligne.

PROCEDURE ReceiveByte(ByteParam:Byte)
Recevoir de la ligne un byte et le mettre dans ByteParam.

VAR BusBusy : BOOLEAN
Indique si une transmission est en cours sur le bus.

2. Interface Liaison de données - Transport

L'interface est constitué de 2 fonctions, de 2 procédures et de 4 variable :

FUNCTION TransmitFrame:Status
Envoyer une trame.
Status indique si la trame a été complètement envoyée ou si la transmission a été avortée pour cause de collision.

PROCEDURE ReceiveFrame
Reception d'une trame complète.
Les trames rentrées en collision ne sont pas délivrées.

PROCEDURE TransmitAcknowledge
Envoyer un acquittement.

FUNCTION ReceiveAcknowledge:Status
Attente d'acquittement.
Status indique si l'acquittement a été reçu, ou si la ligne a été perdue pendant la transaction.

VAR OutgoingFrame, IncomingFrame
Trames à envoyer et reçue.

VAR OutgoingAck, IncomingAck
Acquittement à envoyer ou reçu.

3. Interface Transport - Application

L'interface est constituée de 2 procédures :

```
PROCEDURE SEND(To:AddressValue;
               Length:LengthValue;
               Msg:Message;
               VAR  Code:CodeValue)
```

Envoyer le message, de longueur Length, contenu dans Msg à la station To. Code indique si l'envoi a été fructueux ou non.

```
PROCEDURE RECEIVE (VAR Origin:AddressValue;
                   VAR Length:LengthValue;
                   VAR Msg:Message;
                   VAR Code:CodeValue)
```

Recevoir un message ; mettre ce message dans Msg, sa longueur dans Length et son adresse d'origine dans Origin. Code contient un code d'erreur éventuel.

```
{PHYSICAL LAYER}
{--------------}

{Elements used by the physical layer}

CONST
  ByteSize                        =8;
  TransmittedByteSize             =ByteSize+1 ; {1 Byte + 1 parity bit}

TYPE
  Bit                             =0..1;
  Byte                            =ARRAY [1..ByteSize] OF Bit;
  ParityBit                       =Bit;
  BitPointer                      =1..TransmittedByteSize;

  ByteViewPoint                   =(DataLink,Physical); {A Byte is known from
                                          both Physical & Datalink layers}

  AssembledByte                   =RECORD {Format of an assembled Byte}
                                      CASE View: ByteViewPoint OF
                                        DataLink: (
                                          ByteBody: BYTE;
                                          ByteParity: ParityBit);
                                        Physical: (
                                          Bits: ARRAY [1..TransmittedByteSize]
                                                   OF Bit)
                                      END {AssembledByte};

VAR
  OutgoingByte,IncomingByte       :AssembledByte; {Byte to transmit or to receive}
  CurrentOutgoingBit,CurrentIncomingBit: BitPointer; {Position of current bit
        in Outgoing & Incoming Byte}

  CarrierSense                    :BOOLEAN; {Indicates incoming bits}
  TransmittingByte                :BOOLEAN; {Indicates outgoing bits}
  BusBusy                         :BOOLEAN; {Indicates activity on the bus}
  ByteReady                       :BOOLEAN; {Indicates whether a byte has been
                                             received and is ready for use}
```

```
PROCEDURE TransmitByte (ByteParam: Byte);
{With the given Byte, elaborate the parity bit to obtain an assembled byte,
 then transmit this assembled byte}
BEGIN
  WITH OutgoingByte DO
  BEGIN {Assemble Byte}
    View:=DataLink;
    ByteBody:=ByteParam;
    ByteParity:=Parity (ByteBody) {Elaborate Parity Bit}
  END {Assemble Byte};
  CurrentOutgoingBit:=1;
  TransmittingByte:=TRUE;
END {TransmitByte};



PROCESS BitTransmitter;
{This process wait for the next assembled byte to transmit; when it's
 present, it sends it, bit after bit, on the line}
BEGIN
  CYCLE {Wait for a byte}
    WHILE TransmittingByte DO WITH OutgoingByte DO
    BEGIN {inner loop}
      View:=Physical; {Physical Point of view...}
      TransmitBit (Bits[CurrentOutgoingBit]); {Send next bit on the line}
      NextOutgoingBit {Prepare next bit}
    END {inner loop}
  END {Wait for a byte}
END {BitTransmitter};



PROCEDURE TransmitBit (BitParam: Bit);
{Transmit the given bit on the line : this procedure is able to convert
 logical bits into electrical signals}
BEGIN
  {"0" --> +12V}
  {"1" --> -12V}
END {TransmitBit};



PROCEDURE NextOutgoingBit;
{Prepare next bit;  if no more bits to transmit, stops the transmission}
BEGIN
  CurrentOutgoingBit:=CurrentOutgoingBit+1; {Next Bit}
  TransmittingByte:=(CurrentOutgoingByte<=TransmittedByteSize);
END {NextOutgoingByte};
```

```
PROCEDURE ReceiveByte (ByteParam: Byte);
{Restitute the byte that has been received}
BEGIN
  WHILE NOT ByteReady DO NOTHING; {Wait until a byte is received}

  WITH IncomingByte DO
  BEGIN
    View:=DataLink; {Data Link Point of View ... }
    ByteParam:=ByteBody;
  END;
  ByteReady:=FALSE
END {ReceiveByte};



PROCESS BitReceiver;
{This process detects incoming bits from the line, and try to reassemble
 a byte with them}
BEGIN
  CYCLE {Wait incoming bits}
    WHILE NOT CarrierSense DO NOTHING; {Wait incoming bits}
    CurrentIncomingBit:=1;
    WHILE CarrierSense DO WITH IncomingByte DO
    BEGIN {inner loop}
      View:=Physical; {Physical Point of View ...}
      ReceiveBit (Bits[CurrentIncomingBit]); {Receive the bits from the line}
      NextIncomingBit; {Prepare next incoming bit}
    END {inner loop}
  END {Wait incoming bits}
END {BitReceiver};



PROCEDURE ReceiveBit (BitParam: Bit);
{Receive the bits from the line;  this procedure is able to convert
 electrical signals into logical data bits}
BEGIN
  {-12 --> "1"}
  {+12 --> "0"}
END {ReceiveBit};



PROCEDURE NextIncomingBit;
{Prepare the next incoming bit pointer into incoming byte}
BEGIN
  CurrentIncomingBit:=CurrentIncomingBit+1;
  ByteReady:=(CurrentIncomingBit>9);
END {NextIncomingBit};
```

```
FUNCTION Parity (ByteParam: BYTE): Bit;
{Computes the horizontal parity bit}
VAR i:BitPointer;
BEGIN
  WITH ByteParam DO
  BEGIN {outer loop}
    View:=Physical;
    Parity:=0; {Initializes}
    BEGIN {inner loop}
      FOR i:=1 TO ByteSize DO
        Parity:=Parity XOR bits[i];
    END {inner loop}
  END {outer loop}
END {Parity};


PROCESS BusActivityChecker;
{This process simulates the DSR pins : a timer is set each time a bit is
 received; it's reset some time after if no more bits circulates on the
 line}
VAR Timer: INTEGER; {Microsec. Timer}
BEGIN
  CYCLE
    BusBusy:=FALSE;
    WHILE NOT CarrierSense DO NOTHING; {Wait passing bits...}
    BusBusy:=TRUE;
    WHILE CarrierSense DO NOTHING; {Wait until no more bits circulate}
    Timer:=80; {Initializes the timer to 80 micro. sec}
    WHILE Timer>0 DO Timer:=Timer-1; {Each loop is supposed to hold
        one micro sec.}
  END {CYCLE}
END {BusActicvityChecker};
```

```
{DATA LINK LAYER}
{--------------}

{Elements used by the data link layer}

CONST
  BEL                          =07H;
  ETX                          =03H;
  STX                          =02H;
  ENQ                          =05H
  DataSize                     =1..250; {Data only}
  FrameSize                    =7+DataSize; {DataSize + some control data}

TYPE
  AddressValue                 =Byte;
  LengthValue                  =1..DataSize;
  BytePointer                  =1..FrameSize;

  Status                       =(OK,CAN);

  FrameViewPoint               =(DataLink,Transport); {Frame is known by both
                                  DataLink & Transport layer}
  AckViewPoint                 =(DataLink,Transport); {Same as Frame}

  Frame                        =RECORD {Format of a frame}
                                  CASE View: FrameViewPoint OF
                                    Transport: (
                                      FlagField: BEL;
                                      DestinationField: AddressValue;
                                      SourceField: AddressValue;
                                      TypeField: (STX,ENQ);
                                      LengthField: LengthValue;
                                      DataField: PACKED ARRAY [1..DataSize]
                                                          OF Byte;
                                      BccField: Byte;
                                      EndFlagField: ETX);
                                    DataLink: (
                                      Bytes: ARRAY [1..FrameSize] OF Byte)
                                  END {frame};

  Acknowledge                  = {Acknowledge format}
                                  CASE View: AckViewPoint OF
                                    Transport: (ACK, NAK, DLE);
                                    DataLink: (AckByte: Byte)
                                  END {acknowledge};

  ReceiverProcess              =(Collision,Frame,Dummy); {Process to be
                                   activated when a byte is received}
```

```
VAR
  ByteRead                          :Byte; {Byte that has been read in}
  CollisionByte                     :Byte;

  OutgoingFrame,IncomingFrame       :Frame; {Frame to transmit or to receive}
  OutgoingAck,IncomingAck           :Acknowledge; {Acknowledge to transmit or to
                                          receive}

  CurrentOutgoingByte,CurrentIncomingByte :BytePointer;

  TransmittingFrame,ReceivingFrame :BOOLEAN; {Indicates that a frame is
                                          beeing transmitted or received}
  CollisionDetected                 :BOOLEAN; {indicates that a collision has
                                          been detected}
  NoAcknowledge                     :BOOLEAN; {indicates whether an acknowledge
                                          has been received or not}
  TransmitEnded                     :BOOLEAN; {indicates whether a complete frame
                                          has been sent}

  ReceiverService                   :ReceiverProcess;
```

```
FUNCTION TransmitFrame: Status;
{This function transmit a frame and return the transmission conditions,
 either CAN (Collision detected) or OK (Transmit Ended)}
BEGIN
  WaitBusFree; {Wait until bus becomes free again}
  StartTransmit; {Initializes..}
  WHILE TransmittingFrame DO NOTHING {Wait until Transmitter ends its work};
  TransmitFrame:= IF CollisionDetected
    THEN CAN {Cancelled}
    ELSE OK;
END {TransmitFrame};



PROCEDURE WaitBusFree;
{Listen Before Talk ...}
BEGIN
  WHILE BusBusy DO NOTHING
END {WaitBusFree};



PROCEDURE StartTransmit;
{Do some init. before transmission goes on..}
BEGIN
  CollisionDetected:=FALSE; {Reset}
  ReceiverProcess:=Collision; {Receiver is Collision detector}
  CurrentOutgoingByte:=1;
  CurrentIncomingByte:=1; {Collision Detector purposes}
  TransmitFrame:=TRUE
END {StartTransmit};



PROCESS FrameTransmitter;
{Transmit a frame when one is ready to send}
BEGIN
  CYCLE
    WHILE NOT TransmittingFrame DO NOTHING; {Wait for a frame to send}
    WITH OutgoingFrame DO
    BEGIN {outer loop}
      View:=DataLink; {Disassemble frame}
      WHILE NOT CollisionDetected AND TransmittingFrame DO
      BEGIN {inner loop}
        IF Bytes[CurrentOutgoingByte]<>BEL AND
          Bytes[CurrentOutgoingByte]<>DLE
          THEN  TransmitByte (Bytes[CurrentOutgoingByte]); {Transmit byte
                        after byte}
          ELSE BEGIN {Transparancy control}
            TransmitByte(DLE); {send first the control character}
            IF Bytes[CurrentOutgoingByte]=BEL
              THEN TransmitByte(PAD); {BEL ---> DLE PAD}
              ELSE TransmitByte(DLE); {DLE ---> DLE DLE}
          END {Transparancy control};
        NextOutgoingByte
      END {inner loop}
    END {outer loop}
  END {CYCLE}
END {FrameTransmitter};
```

```
PROCEDURE NextOutgoingByte;
{Prepare next byte to send}
BEGIN
  CurrentOutgoingByte:=CurrentOutgoingByte+1;
  TransmitFrame:=(CurrentOutgoingByte<=FrameSize); {Frame completely transmitted}
END {NextOutgoingByte};


PROCESS CollisionDetector;
{This process detects frame collisions. If one collision is detected when
 frame transmission begins, it stops immediately this transmission.
 Collisions are detected upon the first three bytes of a frame}
BEGIN
  CYCLE
    WHILE ReceiverProcess<>Collision OR CollisionDetected DO NOTHING; {wait
        until a new frame is sent out}
    WITH OutgoingFrame DO
    BEGIN {inner loop}
      view:=DataLink;
      ReceiveByte(CollisionByte); {Listen while talking ...}
      CollisionDetected:=NOT(CollisionByte=Bytes[CurrentIncomingByte]);{Check
                if collision}
      CurrentIncomingByte:=CurrentIncomingByte+1; {Prepare next byte}
      IF CurrentIncomingByte>3 THEN ReceiverProcess:=Dummy; {Disconnect the
                Collision Detector}
    END {inner loop}
  END {CYCLE}
END {CollisionDetector};
```

```
PROCESS ReceiveFrame;
{This process waits for an incoming frame}
BEGIN
  CYCLE
    WaitNewFrame; {wait for a new frame to come in}
    StartReceive; {Initailizes when it does}
    While ReceivingFrame DO NOTHING; {Wait end of frame reception}
  END {CYCLE}
END {ReceiveFrame};



PROCEDURE WaitNewFrame;
{This procedure detects the arriving of a new frame by checking the
 incoming bytes}
BEGIN
  ReceiveByte(ByteRead); {Get a byte}
  WHILE ByteRead<>BEL DO ReceiveByte(ByteRead); {Wait for the BEL flag of
        a new frame to appear}
END {WaitNewFrame};



PROCEDURE StartReceive;
{When a new frame is detected, do some work before threatening with it}
BEGIN
  ReceiverProcess:=Frame; {Frame receiver is active}
  CurrentIncomingByte:=2; {BEL has been allready received}
  ReceivingFrame:=TRUE
END {StartReceive};



PROCESS FrameReceiver;
{Receive the bytes and assemble a frame}
BEGIN
  CYCLE
    WHILE NOT ReceivingFrame DO NOTHING; {Wait for a new frame arrival}
    WITH IncomingFrame DO
    BEGIN {outer loop}
      view:=DataLink; {Assemble Frame}
      WHILE ReceivingFrame DO
      BEGIN {inner loop}
        ReceiveByte(ByteRead); {Get bytes after bytes}
        IF ByteRead=DLE
          THEN BEGIN {Transparancy control}
            ReceiveByte(ByteRead); {DLE either PAD}
            IF ByteRead=PAD THEN ByteRead:=BEL {DLE PAD ---> BEL}
          END {Transparancy control};
        Bytes[CurrentIncomingByte]:=ByteRead;
        NextIncomingByte; {prepare next byte}
      END {inner loop}
    END {outer loop}
  END {CYCLE}
END {FrameReceiver};
```

```
PROCEDURE NextIncomingByte;
BEGIN
  CurrentIncomingByte:=CurrentIncomingByte;
  ReceivingFrame:=(CurrentIncomingByte<=FrameSize); {Check if frame is
        completely received}
  FrameReceived:=CurrentIncomingByte=FrameSize
END {NextIncomingByte};
```

```
PROCEDURE TransmitAcknowledge;
BEGIN
  WITH OutgoingAcknowledge DO
  BEGIN
    View:=DataLink;
    TransmitByte(ACKByte); {Since the acknowledge is one byte composed}
  END
END {TransmitAcknowledge};


FUNCTION ReceiveAcknowledge:Status;
BEGIN
WHILE NOT ByteReady AND BusBusy DO NOTHING;{Wait for an acknowledge or the bus
        coming free again}
IF ByteReady
  THEN WITH IncomingAcknowledge DO
    BEGIN {Ack. present}
      ReceiveAcknowledge:=OK;
      ReceiveByte(ACKByte); {Get the ack.}
    END {Ack. present}
  ELSE {No acknowledge present}
    ReceiveAcknowledge:=CAN
END {ReceiveAcknowledge};
```

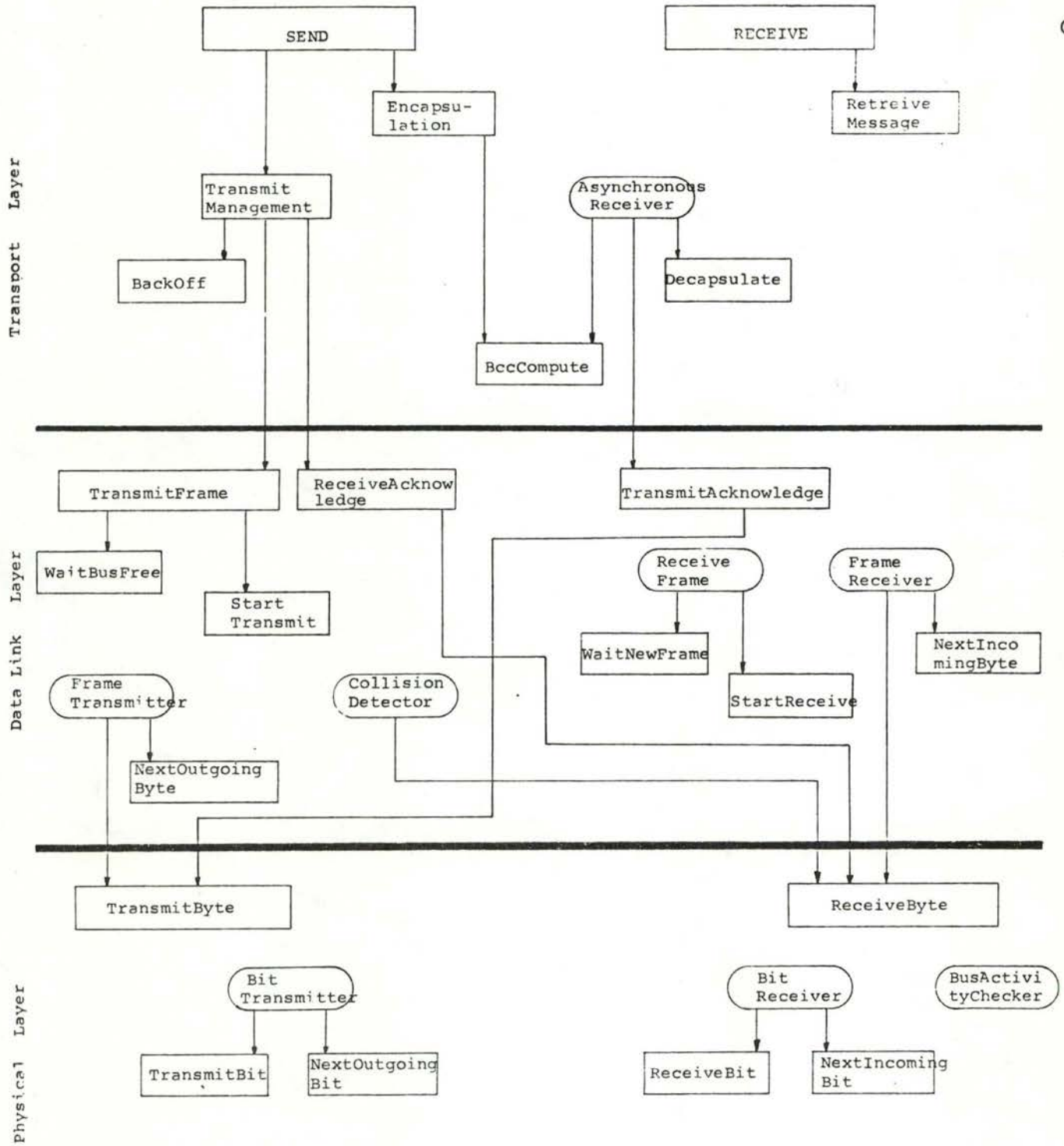
```
{TRANSPORT LAYER}
{--------------}

{Elements used by the transport layer}

CONST
  AttempsLimit                  =5;
  OverallAttempsLimit           =10;
  BasketSize                    = ... {To define};

TYPE
  MESSAGE                       =PACKED ARRAY [1..DataSize] OF Byte;
  Code                          =Byte; {Error purposes}
  Attempts                      =Byte;
  BasketArray                   =PACKED ARRAY [1..BasketSize] OF BYTE;
  BytePointer                   =Byte;

VAR
  CollisionAttempts,TroubleAttempts,SendingAttempts :Attempts;

  HostAddress                   :AdressValue; {Host Specified Address}
  Basket                        :BasketArray;
  BasketEntry, BasketOutry      :BytePointer;
  BasketMessageNumber           :INTEGER;{Number of messages stuffed in the basket}
```

```
PROCEDURE SEND (
        DestinationParam: AddressValue;
        LengthParam: LengthValue;
        MessageParam: Message;
        CodeParam: Code);

VAR Done:BOOLEAN;
BEGIN
  Encapsulation; {Encapsulate Frame}
  TransmitManagement
END {SEND};



PROCEDURE TransmitManagement;
{This procedure takes care of a good and secure transmission of the
 given frame}
BEGIN
  CollisionAttempts:=0; {Attempts reset}
  TroubleAttemps:=0;
  SendingAttempts:=0;
  WHILE NOT Done OR (CollisionAttempts+TroubleAttempts+SendingAttempts
                     <=OverallAttemptsLimit) DO
    CASE TransmitFrame OF
      OK: BEGIN {inner block}
            DO CASE ReceiveAcknowledge; {Receive an Acknowledge}
              OK: DO CASE IncomingAcknowledge OF
                    ACK: BEGIN
                           Done:=TRUE;
                           CodeParam:=0 {No errors}
                         END {ACK};
                    NAK: BEGIN
                           TroubleAttempts:=TroubleAttempts+1;
                           Done:=(TroubleAttempts>AttemptsLimit);
                           CodeParam:=2
                         END {NAK};
                    DLE: BEGIN
                           Done:=TRUE;
                           CodeParam:=4
                         END {DLE};
                  END {IncomingAcknowledge DO Case};
              CAN: BEGIN {No acknowledge received}
                     SendingAttempts:=SendingAttempts+1;
                     Done:=(SendingAttempts>AttemptsLimit);
                     CodeParam:=3
                   END {PAD};
            END {ReceiveAcknowledge DO Case}
          END {inner block};
      CAN: BEGIN
             CollisionAttempts:=CollisionAttempts+1;
             IF CollisionAttempts>AttemptsLimit
               THEN BEGIN
                 Done:=TRUE;
                 CodeParam:=1;
               END
               ELSE Backoff(Collisionattempts); {Wait some time before retrying}
    END {outer DO CASE}
END {TransmitManagement};
```

```
PROCEDURE Encapsulation;
{With the given message, constitute the outgoing frame}
VAR i:INTEGER;
BEGIN
  WITH OugoingFrame DO
  BEGIN {inner loop}
    View:=Transport;
    DestinationField:=DestinationParam;
    SourceField:=HostAddress;
    TypeField:=STX;
    LengthField:=LengthParam;
    FOR i:=1 TO LengthParam DO
      DataField[i]:=MessageParam[i];
    BccField:=BccCompute(OutgoingFrame)
  END {inner loop}
END {Encapsulation};



FUNCTION BccCompute (FrameParam: Frame): Byte;
VAR i:INTEGER;
BEGIN
  BccCompute:=0;
  WITH FrameParam DO
  BEGIN
    View:=DataLink;
    FOR i:=1 TO FrameSize DO
      BccCompute:=BccCompute XOR Bytes[i];
  END
END {BccCompute};



PROCEDURE Backoff (Quantity:INTEGE);
{Schedules a waiting time depending on the quantity given}
BEGIN
  WAIT(SlotTime * Random(0, 2 ^ Quantity))
END {Backoff};



FUNCTION Random (low,high:INTEGER):INTEGER;
BEGIN
  Random:=... {Uniformly distributed random integer r such that low <= r < high}
END {Random};
```

```
PROCEDURE RECEIVE (
        VAR OriginParam: AddressValue;
        VAR LengthParam: LengthValue;
        VAR MessageParam: Message;
        VAR CodeParam: Code);
BEGIN
  WHILE BasketMessageNumber=0 DO NOTHING; {Wait for a message to be present}
  RetreiveMessage {Retreive message from basket}
END {RECEIVE};



PROCEDURE RetreiveMessage;
{This procedure retreive a message from the basket when there is at least
 one available}
VAR i:INTEGER;
BEGIN
  OriginParam:=Basket[NextBasketOutry];
  LengthParam:=Basket[NextBasketOutry]
  FOR i:=1 TO LengthParam DO
    MessageParam[i]:=Basket[NextBasketOutry];
  BasketMessageNumber:=BasketMessageNumber-1;
  BasketFreeSpace:=LengthParam+2; {DataSize+ Length & Origin fields}
END {RetreiveMessage};



PROCESS AsynchronousReceiver;
{This process runs asynchronously with Application Processus;
 It receives messages destinated to the station and save them in the
 basket}
BEGIN
  CYCLE
    WHILE NOT FrameReceived DO NOTHING; {Frame is received at DataLink}
    WITH IncomingFrame DO
      BEGIN {inner loop}
        View:=Transport;
        IF DestinationField=HostAddress {Check if host destinated}
          THEN IF BccCompute(IncomingFrame)=BccField {Check if correctly
                        received}
                THEN IF BasketFreeSpace>=LengthField+2 {Check if enough
                        place in the basket}
                        THEN BEGIN
                          TransmitAcknowledge(ACK);
                          Decapsulate
                        END
                        ELSE TransmitAcknowledge(DLE)
                 ELSE TransmitAcknowledge(NAK)
          ELSE NOTHING; {Frame is not station destinated}
      END {inner loop}
    FrameReceived:=FALSE; {Wait until next frame}
  END {CYCLE}
END {AsynchonousReceiver};
```

```
PROCEDURE Decapsulate;
{Decapsulate the incoming frame}
VAR i:INTEGER
BEGIN
  WITH IncomingFrame DO
    BEGIN
      View:=Transport;
      Basket[NextBasketEntry]:=OriginField;
      Basket[NextBasketEntry]:=LengthField;
      FOR i:=1 TO LengthField DO
        Basket[NextBasketEntry]:=DataField[i];
    END
END {Decapsulate};


FUNCTION NextBasketEntry: BytePointer;
BEGIN
  BasketEntry:=(BasketEntry+1) MOD BasketSize;
  NextBasketEntry:=BasketEntry
END {NextBasketEntry};

FUNCTION NextBasketOutry: BytePointer;
BEGIN
  BasketOutry:=(BasketOutry+1) MOD BasketSize;
  NextBasketOutry:=BasketOutry
END {NextBasketOutry};
```


SEND
RECEIVE
Encapsu-
lation
Retreive
Message
Transport Layer
Transmit
Management
Asynchronous
Receiver
BackOff
Decapsulate
BccCompute
TransmitFrame
ReceiveAcknow
ledge
TransmitAcknowledge
Data Link Layer
WaitBusFree
Start
Transmit
Receive
Frame
Frame
Receiver
WaitNewFrame
NextInco
mingByte
Frame
Transmitter
Collision
Detector
StartReceive
NextOutgoing
Byte
TransmitByte
ReceiveByte
Physical Layer
Bit
Transmitter
Bit
Receiver
BusActivi
tyChecker
TransmitBit
NextOutgoing
Bit
ReceiveBit
NextIncoming
Bit

Annexe C2

Programmes du logiciel de réseau

(Documents confidentiels)

```
OPEN_LINK (VAR host_addr:BYTE;
           VAR err:BYTE);
```

Specifications :

Demande d'accès au réseau.

Cette procédure permet au PA d'accéder au réseau. Elle renvoie au PA appelant l'adresse de la station (host_addr) et un code d'erreur éventuel (err).

Cette procédure vérifie également qu'aucune autre station portant la même adresse ne soit déjà connectée au réseau.

Entrees :

(Rien)

Sorties :
---

host_adress :
contient, après l'appel, l'adresse de la station.

err :
il se peut, dans certains cas, que l'appel à cette procédure ne se soit pas terminé dans les conditions attendues.

Cette variable contient une valeur de terminaison du service :

0 : Pas d'erreur (La liaison est ouverte)
1 : La liaison est déjà ouverte (Elle reste ouverte)
2 : Une autre station est déjà connectée au réseau sous la même adresse (La liaison n'est pas ouverte)
3 : Erreurs répétées lors de l'accès au bus (La liaison n'est pas ouverte).

CLOSE (VAR err:BYTE);

Specifications :

Demande de clotûre d'accès au réseau.

Cette procédure a pour but de fermer l'accès au réseau.
Après l'appel à cette procédure, la station est déconnectée du réseau de sorte que le PA ne puisse plus recevoir de messages en provenance des autres PA qui accèdent au réseau.
Un code d'erreur (err) est retourné au PA, indiquant les conditions de réalisation de cette procédure.

Entree :

(Rien)

Sorties :
---

err :
Il se peut, dans certains cas, que l'appel à cette procédure ne soit pas justifié ; un code d'erreur est alors retourné au programme appelant dans cette variable.

0 : Pas d'erreur (La liaison est fermée)
1 : La liaison était déjà fermée, ou pas encore ouverte, lors de l'appel à cette procédure (La liaison reste fermée)
2 : La liaison est fermée, mais des messages sont encore disponibles dans le panier de réception.

```
SEND    (to_addr:BYTE;
         length:BYTE;
         msg:MESSAGE;
         VAR err:BYTE);
```

Specifications :

Demande d'envoi d'un message.

Cette procédure a pour but d'envoyer un message (msg) dont on connait la longueur (length) à la station dont on spécifie l'adresse (to_addr). Pour des raisons diverses, il se peut que le destinataire n'ait pas reçu le message ou l'ait refusé ; la procédure d'envoi applique la politique du meilleur effort pour assurer un service sûr et fiable au PA utilisateur. Le PA peut savoir dans quelles conditions l'envoi s'est réalisé ; dans ce but, la procédure renvoie au PA un code d'erreur (err).

Entrees :
---

to_addr :
Adrese de la station à laquelle le message doit être envoyé. Cette adresse peut prendre n'importe quelle valeur de l'intervalle [1..255], pourvu que ce ne soit pas l'adresse de la station émettrice.

length :
Longueur du message à envoyer. Cette longueur, calculée en nombre de bytes, peut prendre n'importe quelle valeur de l'interva [1..250].

msg :
C'est le message à envoyer.
Ce message, constitué par un PACKED ARRAY [1..250] OF BYTE, comprend les données à transmettre.

Sorties :
---

err :
Il se peut néanmoins, qu'après plusieurs essais, le message n'ait pas pu être convenablement acheminé vers la station destinatrice. Un code de réalisation de la procédure est renvoyé dans cette variable.

Ce code est, suivant le cas :

0 : Pas d'erreur (Le message a été accepté par la station destinatrice).
1 : Trop d'essais - collision ; le message est entré plusieurs fois en collision avec un autre message.
2 : Trop d'essais - parasites; à plusieurs reprises, le message a été mal reçu par la station destinatrice.
3 : Trop d'essais - station inexistante; le message a été envoyé plusieurs fois mais aucun acquittement en provenance de la station destinatrice n'a été reçu (Cette station semble absente).
4 : Station destinatrice pas prête : le message a été convenablement reçu par la station destinatrice, mais celle-ci ne dispose plus d'assez de place pour stocker ce message.
5 : Erreur inconnue : une erreur inconnue est apparue en cours de transmission.
6 : Adresse de destination invalide : l'adresse de destination est invalide (0 ou adresse de la station même).
7 : Longueur du message invalide (0 ou > 250).
8 : Accès au réseau non autorisé.

```
RECEIVE (VAR from_addr:BYTE;
         VAR length:BYTE;
         VAR message:MESSAGE;
         wait: BOOLEAN;
         VAR err:BYTE);
```

Specifications :

Demande de réception de message.

Deux solutions sont possibles lors de l'appel à cette procédure. Dans la première solution (wait=vrai), un message peut être déjà disponible, auquel cas la procédure renvoie ce message (msg), sa longueur (lengt) et l'adresse de son expéditeur (from_addr). Si aucun message n'est disponible lors de l'appel à cette procédure, le PA est bloqué jusqu'à disponibilité d'un nouveau message. Dans la seconde solution (wait=faux), si aucun message n'est disponible, le PA n'est pas bloqué et un code d'erreur (err) est renvoyé. Si un message est disponible, il est fourni au PA comme dans le cas précédent.

N.B. : Un PA peut faire appel à cette procédure même après clotûre de l'accès au réseau, pour prendre les messages restant qui ont été reçus avant cette clotûre.

Entree :
---

wait :
Cette variable est utilisée pour spécifier si on désire attendre au cas où aucun message ne serait disponible.

TRUE : Attente si aucun message n'est disponible.
FALSE : Pas d'attente ; si aucun message n'est disponible, un code d'erreur est renvoyé.

Sorties :
---

from_addr :
Adresse de la station expéditrice du message reçu.

length :
Longueur du message de données reçu.

msg :

Le message de donnèes lui-même.
Il s'agit d'un tableau de bytes, PACKED ARRAY [1..250] OF BYTE.

err :

Un code d'erreur èventuel renseigne des conditions de rèalisation de cette procè-dure :

0 : Pas d'erreur (Un message est renvoyè dans msg).
1 : Pas de message disponible (Option "No wait").
8 : Pas de message disponible (Accès au rèseau fermè).

ANNEXE D

Divers

HEX				MSD	p = 1	8	9	A	B	C	D	E	F
					p = 0	0	1	2	3	4	5	6	7
	BITS				b8	p	p	p	p	p	p	p	p
					b7	0	0	0	0	1	1	1	1
					b6	0	0	1	1	0	0	1	1
LSD	b4	b3	b2	b5 / b1		0	1	0	1	0	1	0	1
0	0	0	0	0		NUI	DLE	SP	0	@	P	`	p
1	0	0	0	1		SOH	DCI	!	1	A	Q	a	q
2	0	0	1	0		STX	DC2	"	2	B	R	b	r
3	0	0	1	1		ETX	DC3	#	3	C	S	c	s
4	0	1	0	0		EOT	DC4	$	4	D	T	d	t
5	0	1	0	1		ENQ	NAK	%	5	E	U	e	u
6	0	1	1	0		ACK	SYN	&	6	F	V	f	v
7	0	1	1	1		BEL	ETB	'	7	G	W	g	w
8	1	0	0	0		BS	CAN	(	8	H	X	h	x
9	1	0	0	1		HT	EM	)	9	I	Y	i	y
A	1	0	1	0		LF	SUB	*	:	J	Z	j	z
B	1	0	1	1		VT	ESC	+	;	K	[	k	{
C	1	1	0	0		FF	FS	,	<	L	\	l	|
D	1	1	0	1		CR	GS	-	=	M	]	m	}
E	1	1	1	0		SO	RS	.	>	N	^	n	~
F	1	1	1	1		SI	US	/	?	O	_	o	DEL

CONTROL CHARACTERS

NUL	Null	**FF**	Form Feed	**CAN**	Cancel
SOH	Start of Heading	**CR**	Carriage Return	**EM**	End of Medium
STX	Start of Text	**SO**	Shift Out	**SUB**	Substitute
ETX	End of Text	**SI**	Shift In	**ESC**	Escape
EOT	End of Transmission	**DLE**	Data Link Escape	**FS**	File Separator
ENQ	Enquiry	**DC1**	Device Control 1	**GS**	Group Separator
ACK	Acknowledge	**DC2**	Device Control 2	**RS**	Record Separator
BEL	Bell (audible or attention signal)	**DC3**	Device Control 3	**US**	Unit Separator
BS	Backspace	**DC4**	Device Control 4 (Stop)	**DEL**	Delete
HT	Horizontal Tabulation (punched card skip)	**NAK**	Negative Acknowledge		
LF	Line Feed	**SYN**	Synchronous Idle		
VT	Vertical Tabulation	**ETB**	End of Transmission Block		

ASCII Code Assignments

DESCRIPTION DE LA CARTE ISBC 544

Il s'agit d'un interface intelligent, i.e. programmable, composé de plusieurs circuits de contrôle dont les trois suivants retiennent principalement notre attention :

1. Un interface série de communication programmable INTEL 8251 dont les fonctions permettent suivant sa programmation :

   - d'assurer des opérations en mode synchrone ou asynchrone, et full duplex;

   - en mode synchrone :

     - de manipuler des caractères de 5 à 8 bits;
     - d'utiliser un caractère de synchronisation interne ou externe (mode chasse);
     - d'insérer automatiquement des caractères de synchronisation;
     - de définir une vitesse de transmission jusqu'à 64Kbauds;

   - en mode asynchrone :

     - de manipuler des caractères de 5 à 8 bits;
     - de définir le nombre de start- et stop-bits (1, 1.5 ou 2);
     - de définir une vitesse de transmission jusqu'à 19.200 bauds;

   - de détecter les erreurs de parité, d'engorgement (Overrun) ou de trame (Framing).

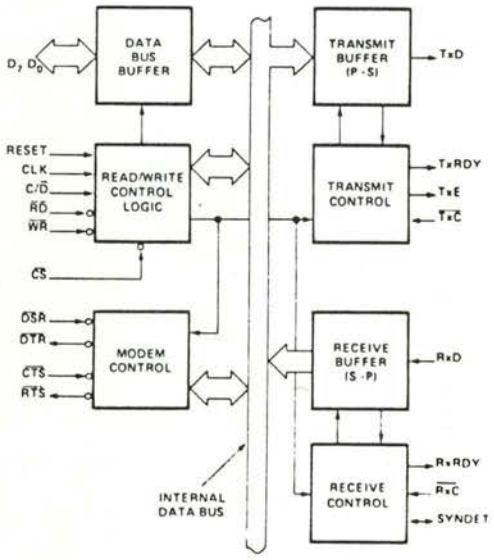


Fig. 1 Interface de communication programmable (8251).

Il peut être entièrement controlé par le microprocesseur 8080 auquel il est raccordé via les bus d'adresse et de données, grâce à certains signaux de contrôle (Fig. 1).

2. Un <u>controleur d'horloge</u> INTEL 8253 (Programmable Interval Timer ou PIT) qui permet de résoudre la plupart des problèmes de gestion d'horloge que l'on rencontre dans les micro-ordinateurs. Au lieu d'insérer des boucles de délais dans le software de base, le programmeur configure ce circuit selon ses exigences, initialise un des trois compteurs (Timer) qui sur commande laissera s'écouler le temps introduit, avant d'interrompre le processeur pour lui signifier la fin de sa tâche. On remarque aisément le soulagement introduit au niveau du software.

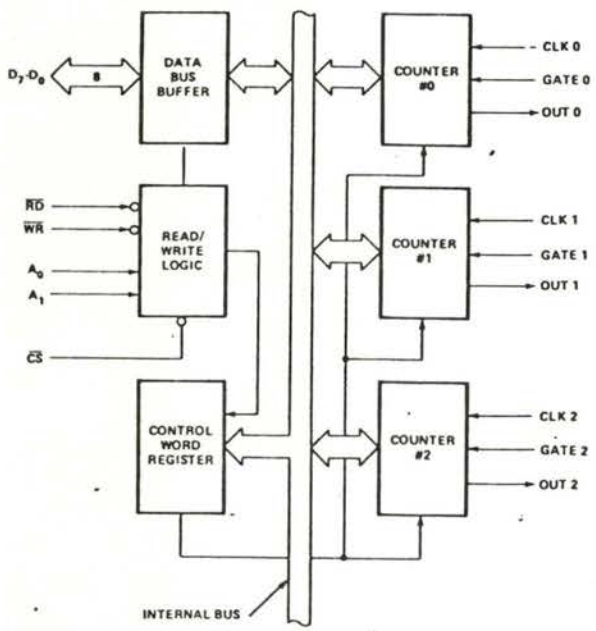


Fig. 2 Controlleur d'horloge programmable (8253).

D'autres fonctions que les délais peuvent également être programmées :

- générateur programmable;
- compteur d'évènements;
- horloge temps réel;
- controleur de moteur, etc.

3. Un <u>controleur d'interruption</u> INTEL 8259 (Programmable Interrupt controller ou PIC) :

<u>fonctions du PIC</u> :

Le PIC fonctionne en temps que controleur principal dans un environnement de système piloté par interruption (Interrupt driven). Il accepte les requêtes des équipements périphériques, détermine parmi

celles qui arrivent laquelle a le plus d'importance (Priorité), s'assure que la requête qui arrive est de priorité plus élevée que celle servie à ce moment-là et emet, vers le CPU, une interruption basée sur cet évènement.

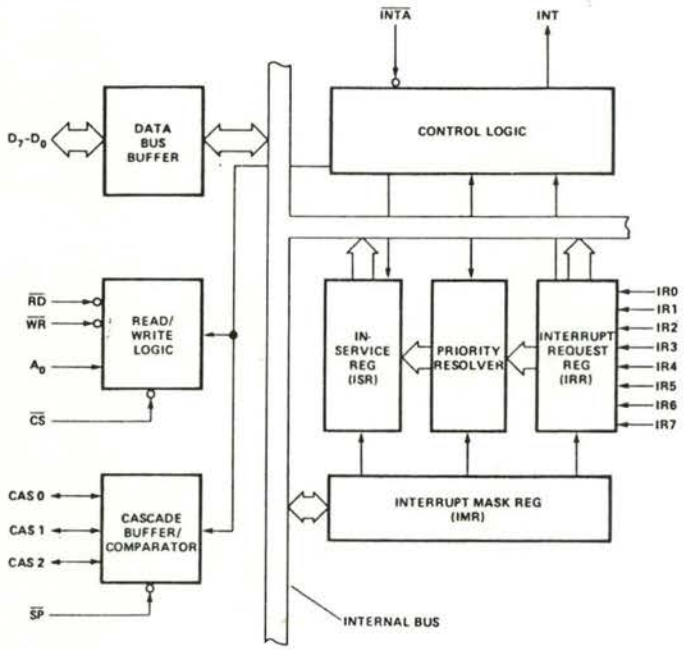


Fig. 3 Controlleur d'interruption programmable (8259).

Chacun des périphériques est controlé par un programme, ou routine, qui lui est associé ainsi que ses spécifications fonctionnelles ou ses exigeances d'opération Le controleur, après avoir interrompu le processeur, doit en outre lui donner certaines informations qui lui permettront de se "diriger" vers la routine de service correspondante. Le controleur fait celà en fournissant au CPU l'adresse de la routine à exécuter.

Le PIC permet en outre :

- de gérer de 8 à 64 niveaux d'interruption;
- de régir les priorités grâce à un masque d'interruption.

Utilité d'un tel circuit :

L'architecture des système de micro-ordinateurs exige que les périphériques d'entrées-sorties -comme les claviers, les écrans, senseurs et autres composants- reçoivent le service du processeur selon une méthode efficace de telle sorte que les tâches du système puissent être exécutées par le micro-ordinateur avec peu ou pas

d'effet sur son rendement.

La méthode la plus commune pour servir de tels périphériques est l'approche par scrutation (Polling) : le processeur teste chacun des périphériques les uns après les autres pour voir lequel a besoin de son service. On remarque vite que la majeure partie du programme principal "boucle" au travers de ce scrutage de périphériques, de sorte que cette méthode aurait un effet néfaste sur le rendement du système, limitant ainsi les tâches assurées par le micro-ordinateur.

Une méthode plus avantageuse serait d'autoriser le microprocesseur à exécuter son programme principal en ne s'arrêtant que pour servir le périphérique qui en aurait fait explicitement la demande. En fait, cette méthode provoquerait une demande asynchrone provenant de l'extérieur qui informerait le processeur d'exécuter une routine qui puisse servir le périphérique qui a fait la requête après avoir terminé l'instruction en cours. Ce service accompli, le processeur revient là où il avait abandonné son programme principal lors du déroutement.

C'est le principe d' interruption. Cette méthode accroît de manière considérable le rendement du système.

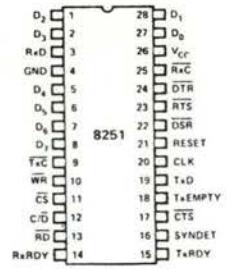
intel

8251
PROGRAMMABLE COMMUNICATION INTERFACE

- **Synchronous and Asynchronous Operation**
  - **Synchronous:**
    5-8 Bit Characters
    Internal or External Character Synchronization
    Automatic Sync Insertion
  - **Asynchronous:**
    5-8 Bit Characters
    Clock Rate — 1, 16 or 64 Times Baud Rate
    Break Character Generation
    1, 1½, or 2 Stop Bits
    False Start Bit Detection
- **Baud Rate — DC to 56k Baud (Sync Mode)**
  DC to 9.6k Baud (Async Mode)
- **Full Duplex, Double Buffered, Transmitter and Receiver**
- **Error Detection — Parity, Overrun, and Framing**
- **Fully Compatible with 8080 CPU**
- **28-Pin DIP Package**
- **All Inputs and Outputs Are TTL Compatible**
- **Single 5 Volt Supply**
- **Single TTL Clock**

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Chip designed for data communications in microcomputer systems. The USART is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM Bi-Sync). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPT. The chip is constructed using N-channel silicon gate technology.

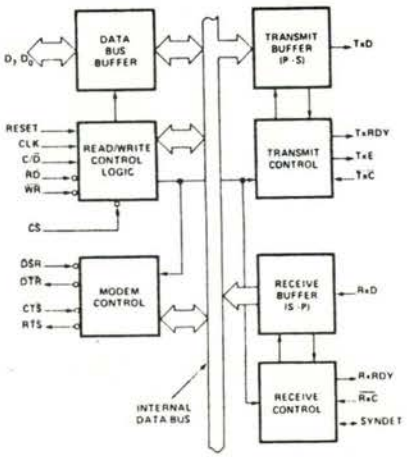
PIN CONFIGURATION



Pin Name	Pin Function
D_7 D_0	Data Bus (8 bits)
C/D	Control or Data is to be Written or Read
RD	Read Data Command
WR	Write Data or Control Command
CS	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
TxC	Transmitter Clock
TxD	Transmitter Data
RxC	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for char. from 8080)

Pin Name	Pin Function
DSR	Data Set Ready
DTR	Data Terminal Ready
SYNDET	Sync Detect
RTS	Request to Send Data
CTS	Clear to Send Data
TxE	Transmitter Empty
V_{CC}	+5 Volt Supply
GND	Ground

BLOCK DIAGRAM



8251 BASIC FUNCTIONAL DESCRIPTION

General

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter designed specifically for the 8080 Microcomputer System. Like other I/O devices in the 8080 Microcomputer System its functional configuration is programmed by the systems software for maximum flexibility. The 8251 can support virtually any serial data technique currently in use (including IBM "bi-sync").

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8251 to the 8080 system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the 8080 CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer.

Read/Write Control Logic

This functional block accepts inputs from the 8080 Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for device functional definition.

RESET (Reset)

A "high" on this input forces the 8251 into an "Idle" mode. The device will remain at "Idle" until a new set of control words is written into the 8251 to program its functional definition. Minimum RESET pulse width is 6 t_{CY}.

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the 8224 Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter clock inputs for synchronous mode (4.5 times for asynchronous mode).

$\overline{WR}$ (Write)

A "low" on this input informs the 8251 that the CPU is outputting data or control words, in essence, the CPU is writing out to the 8251.

$\overline{RD}$ (Read)

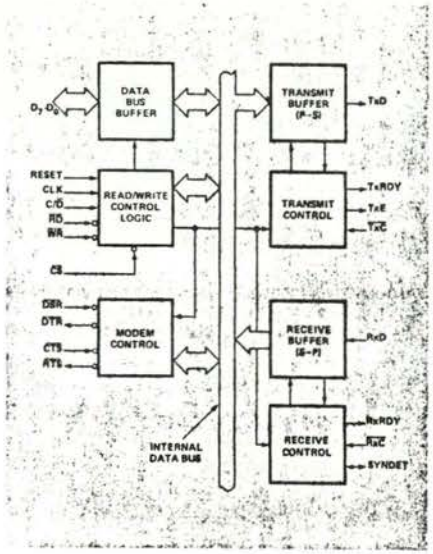
A "low" on this input informs the 8251 that the CPU is inputting data or status information, in essence, the CPU is reading from the 8251.

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs informs the 8251 that the word on the Data Bus is either a data character, control word or status information.
1 = CONTROL 0 = DATA

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8251. No reading or writing will occur unless the device is selected.



C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251 → DATA BUS
0	1	0	0	DATA BUS → 8251
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	1	1	0	DATA BUS → 3-STATE
X	X	X	1	DATA BUS → 3-STATE

Modem Control

The 8251 has a set of control inputs and outputs that can be used to simplify the interface to almost any Modem. The modem control signals are general purpose in nature and can be used for functions other than Modem control, if necessary.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input signal is general purpose in nature. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test Modem conditions such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready)

The $\overline{DTR}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{DTR}$ output signal is normally used for Modem control such as Data Terminal Ready or Rate Select.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{RTS}$ output signal is normally used for Modem control such as Request to Send.

$\overline{CTS}$ (Clear to Send)

A "low" on this input enables the 8251 to transmit data (serial) if the Tx EN bit in the Command byte is set to a "one."

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

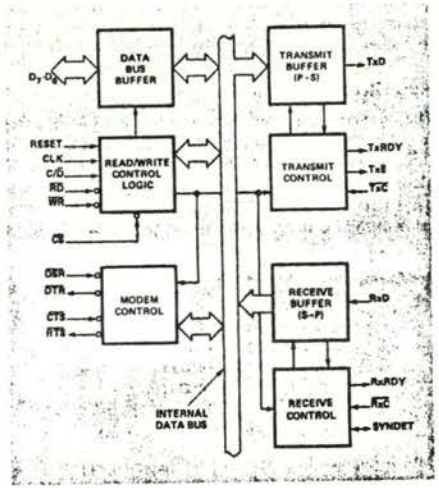
TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. It can be used as an interrupt to the system or for the Polled operation the CPU can check TxRDY using a status read operation. TxRDY is automatically reset when a character is loaded from the CPU.

TxE (Transmitter Empty)

When the 8251 has no characters to transmit, the TxE output will go "high". It resets automatically upon receiving a character from the CPU. TxE can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplexed operational mode. TxE is independent of the TxEN bit in the Command instruction.

In SYNChronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be transmitted automatically as "fillers". TxE goes low as soon as the SYNC is being shifted out.



$\overline{TxC}$ (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the frequency of $\overline{TxC}$ is equal to the actual Baud Rate (1X). In Asynchronous transmission mode, the frequency of $\overline{TxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1x, 16x or 64x the Baud Rate.

For Example:

If Baud Rate equals 110 Baud,
$\overline{TxC}$ equals 110 Hz (1x)
$\overline{TxC}$ equals 1.76 kHz (16x)
$\overline{TxC}$ equals 7.04 kHz (64x).

The falling edge of $\overline{TxC}$ shifts the serial data out of the 8251.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to the RxD pin.

Receiver Control

This functional block manages all receiver-related activities.

RxRDY (Receiver Ready)

This output indicates that the 8251 contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or for Polled operation the CPU can check the condition of RxRDY using a status read operation. RxRDY is automatically reset when the character is read by the CPU.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the frequency of $\overline{RxC}$ is equal to the actual Baud Rate (1x). In Asynchronous Mode, the frequency of $\overline{RxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1x, 16x or 64x the Baud Rate.

For Example: If Baud Rate equals 300 Baud,
$\overline{RxC}$ equals 300 Hz (1x)
$\overline{RxC}$ equals 4800 Hz (16x)
$\overline{RxC}$ equals 19.2 kHz (64x).
If Baud Rate equals 2400 Baud,
$\overline{RxC}$ equals 2400 Hz (1x)
$\overline{RxC}$ equals 38.4 kHz (16x)
$\overline{RxC}$ equals 153.6 kHz (64x).

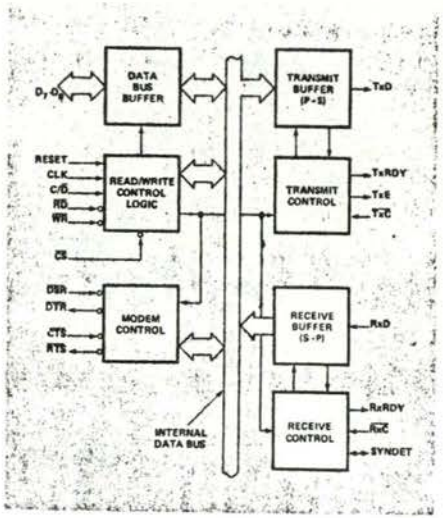
Data is sampled into the 8251 on the rising edge of $\overline{RxC}$.

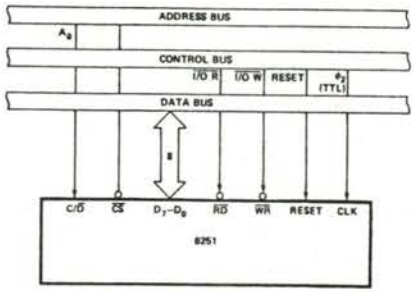
NOTE: In most communications systems, the 8251 will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

SYNDET (SYNC Detect)

This pin is used in SYNChronous Mode only. It is used as either input or output, programmable through the Control Word. It is reset to "low" upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251 has located the SYNC character in the Receive mode. If the 8251 is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input, (external SYNC detect mode), a positive going signal will cause the 8251 to start assembling data characters on the falling edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. The duration of the high signal should be at least equal to the period of $\overline{RxC}$.





8251 Interface to 8080 Standard System Bus

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize the 8251 to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD PARITY etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251 is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251 is ready to receive a character. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251. On the other hand, the 8251 receives serial data from the MODEM or I/O device, upon receiving an entire character the RxRDY output is raised "high" to signal the CPU that the 8251 has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU read operation.

The 8251 cannot begin transmission until the TxEN(Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send (CTS) input. The TxD output will be held in the marking state upon Reset.

Programming the 8251

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

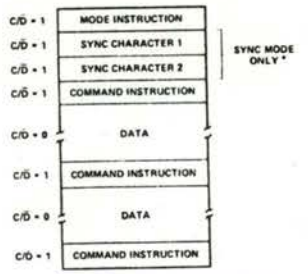
This format defines the general operational characteristics of the 8251. It must follow a Reset operation (internal or external). Once the Mode instruction has been written into the 8251 by the CPU, SYNC characters or Command instructions may be inserted.

Command Instruction

This format defines a status word that is used to control the actual operation of the 8251.

Both the Mode and Command instructions must conform to a specified sequence for proper device operation. The Mode Instruction must be inserted immediately following a Reset operation, prior to using the 8251 for data communication.

All control words written into the 8251 after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251 at any time in the data block during the operation of the 8251. To return to the Mode Instruction format a bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251 back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.



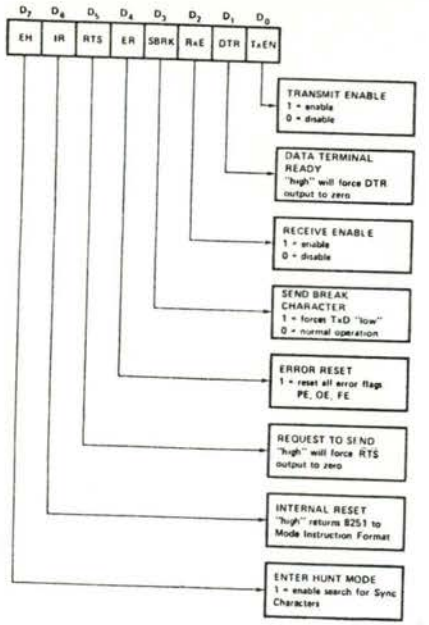
*The second SYNC character is skipped if MODE instruction has programmed the 8251 to single character Internal SYNC Mode. Both SYNC characters are skipped if MODE instruction has programmed the 8251 to ASYNC mode

Typical Data Block

COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251 has been programmed by the Mode Instruction and the Sync Characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

Once the Mode Instruction has been written into the 8251 and Sync characters inserted, if necessary, then all further "control writes" (C/$\overline{D}$ = 1) will load the Command Instruction. A Reset operation (internal or external) will return the 8251 to the Mode Instruction Format.



Command Instruction Format

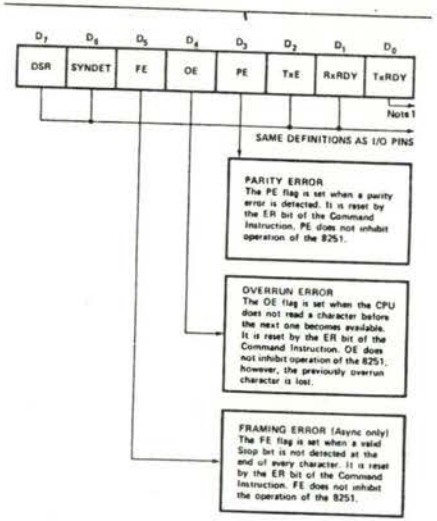
STATUS READ DEFINITION

In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251 has facilities that allow the programmer to "read" the status of the device at any time during the functional operation.

A normal "read" command is issued by the CPU with the C/D input at one to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251 can be used in a completely Polled environment or in an interrupt driven environment.

Status update can have a maximum delay of 16 clock periods.



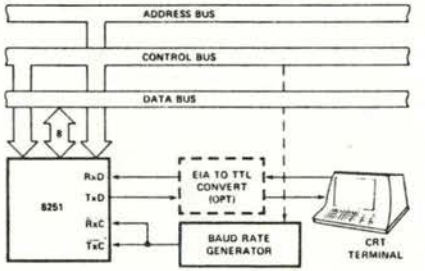
Status Read Format

Note 1: TxRDY status bit has similar meaning as the TxRDY output pin. The former is not conditioned by CTS and TxEN; the latter is conditioned by both CTS and TxEN.

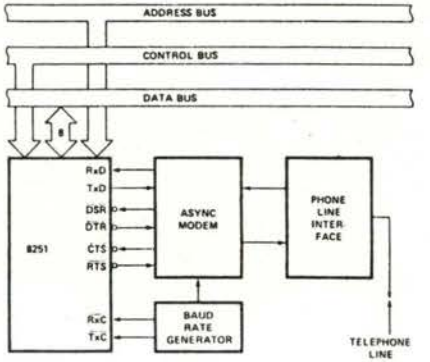
i.e. TxRDY status bit = DB Buffer Empty

TxRDY pin out = DB Buffer Empty • CTS • TxEN

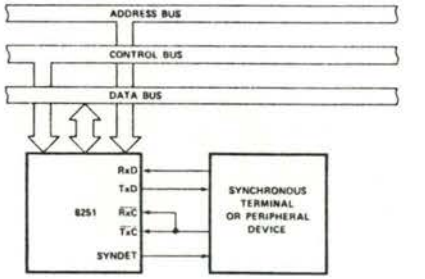
APPLICATIONS OF THE 8251



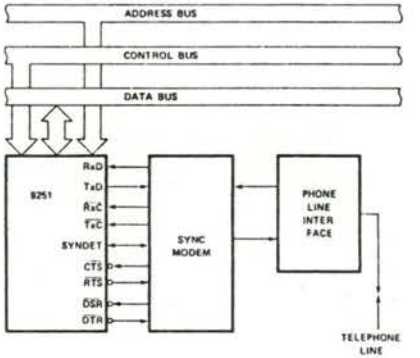
Asynchronous Serial Interface to CRT Terminal, DC-9600 Baud



Asynchronous Interface to Telephone Lines



Synchronous Interface to Terminal or Peripheral Device



Synchronous Interface to Telephone Lines

Absolute Maximum Ratings*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . –65°C to +150°C
Voltage On Any Pin
With Respect to Ground . . . . . –0.5V to +7V
Power Dissipation . . . . . 1 Watt

COMMENT: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. Characteristics:

T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -100μA
I_{DL}	Data Bus Leakage			-50	μA	V_{OUT} = .45V
				10	μA	V_{OUT} = V_{CC}
I_{IL}	Input Leakage			10	μA	V_{IN} = V_{CC}
I_{CC}	Power Supply Current		45	80	mA	

Capacitance:

T_A = 25°C; V_{CC} = GND = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND.

TEST LOAD CIRCUIT:

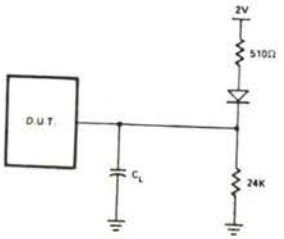
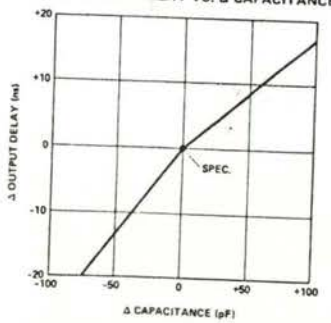


Figure 1.

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE (dB)



A.C. Characteristics:

T_A = 0°C to 70°C; V_{CC} = 5.0V ±5%; GND = 0V

BUS PARAMETERS: (Note 1)

READ CYCLE

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AR}	Address Stable Before $\overline{READ}$ ($\overline{CS}$, C/D)	50		ns	
t_{RA}	Address Hold Time for $\overline{READ}$ ($\overline{CS}$, C/D)	5		ns	
t_{RR}	$\overline{READ}$ Pulse Width	430		ns	
t_{RD}	Data Delay from $\overline{READ}$		350	ns	C_L = 100 pF
t_{DF}	$\overline{READ}$ to Data Floating		200	ns	C_L = 100 pF
		25		ns	C_L = 15 pF
t_{RV}	Recovery Time Between WRITES (Note 2)	6		t_{CY}	

WRITE CYCLE

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{AW}	Address Stable Before $\overline{WRITE}$	20		ns	
t_{WA}	Address Hold Time for $\overline{WRITE}$	20		ns	
t_{WW}	$\overline{WRITE}$ Pulse Width	400		ns	
t_{DW}	Data Set Up Time for $\overline{WRITE}$	200		ns	
t_{WD}	Data Hold Time for $\overline{WRITE}$	40		ns	

NOTES: 1. AC timings measured at V_{OH} = 2.0, V_{OL} = .8, and with load circuit of Figure 1.
2. This recovery time is for initialization only, when MODE, SYNC1, SYNC2, COMMAND and first DATA BYTES are written into the USART. Subsequent writing of both COMMAND and DATA are only allowed when TxRDY = 1.

Mode Instruction Definition

The 8251 can be used for either Asynchronous or Synchronous data communication. To understand how the Mode Instruction defines the functional operation of the 8251 the designer can best view the device as two separate components sharing the same package. One Asynchronous the other Synchronous. The format definition can be changed "on the fly" but for explanation purposes the two formats will be isolated.

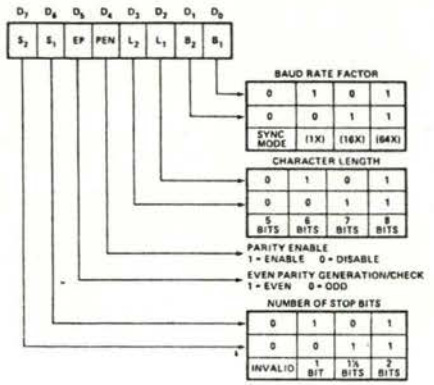
Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251 automatically adds a Start bit (low level) and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, 1/16, or 1/64 that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

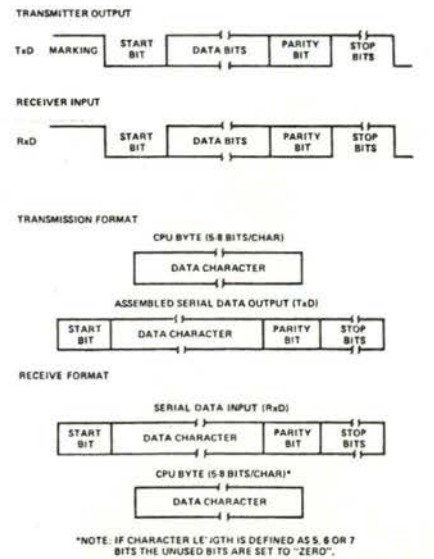
When no data characters have loaded into the 8251 the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center. If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter locates the center of the data bits, the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of $\overline{RxC}$. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. This character is then loaded into the parallel I/O buffer of the 8251. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN flag is raised (thus the previous character is lost). All of the error flags can be reset by a command instruction. The occurrence of any of these errors will not stop the operation of the 8251.



Mode Instruction Format, Asynchronous Mode

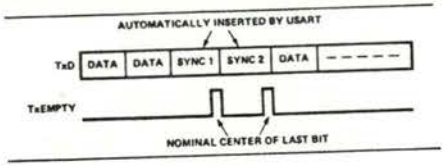


Asynchronous Mode

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251 which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251 with a character before the 8251 becomes empty, the SYNC characters (or character if in single SYNC word mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251 is empty and SYNC characters are being sent out. TxEMPTY goes low when SYNC is being shifted out (See Figure below). The TxEMPTY pin is internally reset by the next character being written into the 8251.



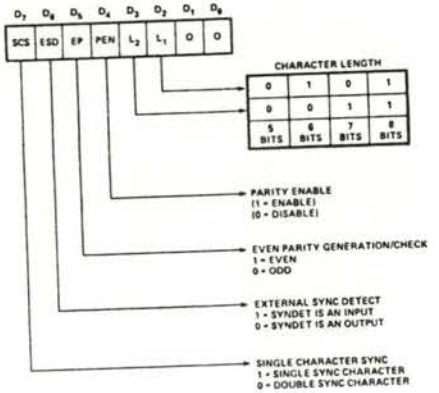
Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the internal SYNC mode has been programmed, the receiver starts in a HUNT mode. Data on the RxD pin is then sampled in on the rising edge of $\overline{RxC}$. The content of the Rx buffer is continuously compared with the first SYNC character until a match occurs. If the 8251 has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ.

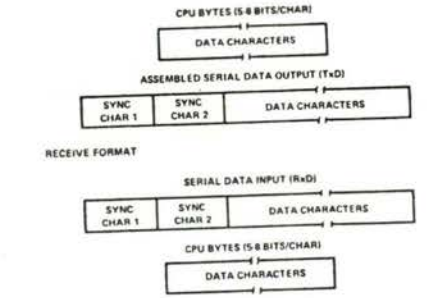
In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin. The high level can be removed after one $\overline{RxC}$ cycle.

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost.



Mode Instruction Format, Synchronous Mode



Synchronous Mode, Transmission Format

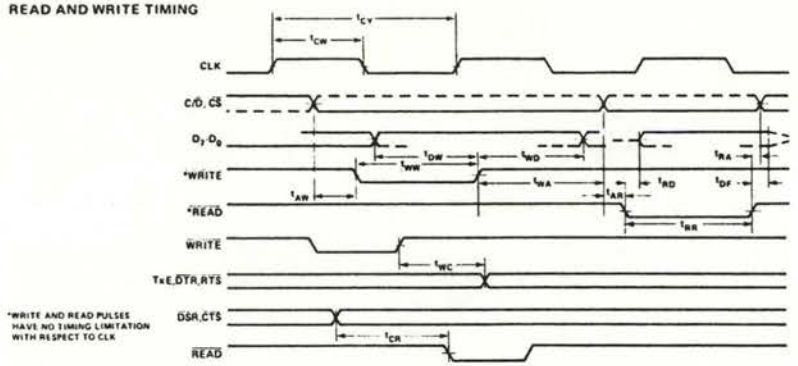
OTHER TIMINGS:

SYMBOL	PARAMETER	MIN.	MAX.	UNIT	TEST CONDITIONS
t_{CY}	Clock Period (Note 3)	.420	1.35	μs	
$t_{\phi W}$	Clock Pulse Width	220	.7 t_{CY}	ns	
t_R, t_F	Clock Rise and Fall Time	0	50	ns	
t_{DTx}	TxD Delay from Falling Edge of TxC		1	μs	C_L = 100 pF
t_{SRx}	Rx Data Set-Up Time to Sampling Pulse	2		μs	C_L = 100 pF
t_{HRx}	Rx Data Hold Time to Sampling Pulse	2		μs	C_L = 100 pF
f_{Tx}	Transmitter Input Clock Frequency				
	1x Baud Rate	DC	56	KHz	
	16x and 64x Baud Rate	DC	520	KHz	
t_{TPW}	Transmitter Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{TPD}	Transmitter Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
f_{Rx}	Receiver Input Clock Frequency				
	1x Baud Rate	DC	56	KHz	
	16x and 64x Baud Rate	DC	520	KHz	
t_{RPW}	Receiver Input Clock Pulse Width				
	1x Baud Rate	12		t_{CY}	
	16x and 64x Baud Rate	1		t_{CY}	
t_{RPD}	Receiver Input Clock Pulse Delay				
	1x Baud Rate	15		t_{CY}	
	16x and 64x Baud Rate	3		t_{CY}	
t_{Tx}	TxRDY Delay from Center of Data Bit		16	t_{CY}	C_L = 50 pF
t_{Rx}	RxRDY Delay from Center of Data Bit		20	t_{CY}	
t_{IS}	Internal SYNDET Delay from Center of Data Bit		25	t_{CY}	
t_{ES}	Internal SYNDET Set-Up Time Before Falling Edge of RxC		16	t_{CY}	
t_{TxE}	TxEMPTY Delay from Center of Data Bit		16	t_{CY}	C_L = 50 pF
t_{WC}	Control Delay from Rising Edge of WRITE (TxE,$\overline{DTR}$,$\overline{RTS}$)		16	t_{CY}	
t_{CR}	Control to READ Set-Up Time ($\overline{DSR}$,$\overline{CTS}$)		16	t_{CY}	

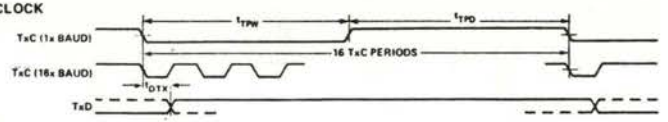
3. The TxC and RxC frequencies have the following limitations with respect to CLK.
   For 1x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(30\ t_{CY})$
   For 16x and 64x Baud Rate, f_{Tx} or $f_{Rx} \leq 1/(4.5\ t_{CY})$

4. Reset Pulse Width = 6 t_{CY} minimum.

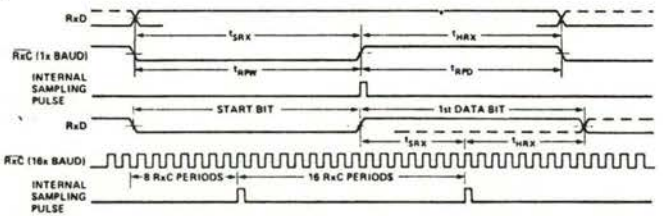
READ AND WRITE TIMING



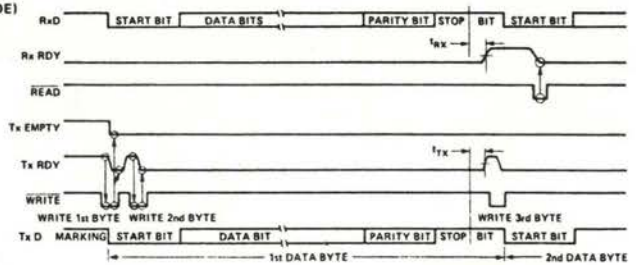
TRANSMITTER CLOCK AND DATA



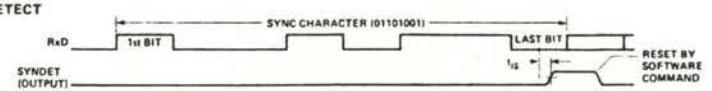
RECEIVER CLOCK AND DATA



Tx RDY AND Rx RDY TIMING (ASYNC MODE)



INTERNAL SYNC DETECT



EXTERNAL SYNC DETECT